NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DEVELOPMENT OF A UAV TRACK INJECTION AND IMAGERY PRESENTATION SYSTEM

by

Andrew R. Cameron
John D. Cherry

March 1999

Thesis Advisors: John Osmundson
Gary Porter

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1999	Master's Thesis

4. TITLE AND SUBTITLE
Development of a UAV Track Injection and Imagery Presentation System

5. FUNDING NUMBERS

6. AUTHOR(S)
Andrew R. Cameron, John D. Cherry

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The Navy is increasingly using advanced Unmanned Air Vehicles (UAVs) to perform critical missions. UAVs have grown in capability, while the Navy's underlying Command and Control structure has changed little to take advantage of the advances in technology. While UAVs are rapidly developing the potential to be effective combat tools, learning how to utilize this potential in an integrated Command and Control environment is hampered by a lack of UAV connectivity.

This thesis develops a methodology for using UAV telemetry data packets to inject tracks of the UAV into a Command and Control system such as the Global Command and Control System (GCCS), and provide near-real-time imagery delivery from the UAV to tactical end users via a network such as the Secret Internet Protocol Router Network (SIPRNET). Focus is on the development of a proof-of-concept system utilizing the Naval Postgraduate School's Systems Technology Battle Lab (STBL) and the Center for Interdisciplinary Remotely-Piloted Aircraft Studies (CIRPAS) Altus UAV. Through the developing of this system, the Altus UAV can serve as a research tool for further development of Command and Control doctrine for operational UAVs.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Unmanned Air Vehicle, Track Injection, Telemetry, Imagery Presentation, Global Command and Control System, Center for Interdisciplinary Remotely-Piloted Aircraft Studies, Altus	199
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFI- CATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

DEVELOPMENT OF A UAV TRACK INJECTION AND IMAGERY PRESENTATION SYSTEM

Andrew R. Cameron
Lieutenant Commander, United States Navy
B.A., University of California Berkeley, 1987

John D. Cherry
Lieutenant, United States Navy
B.S., University of Nebraska Lincoln, 1988

Submitted in partial fulfillment of the
Requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
March 1999

Author: _____
Andrew R. Cameron

Author: _____
John D. Cherry

Approved by: _____
John Osmundson, Thesis Advisor

_____
Gary Porter, Thesis Advisor

_____
Reuben Harris, Chairman
Department of System Management

ABSTRACT

The Navy is increasingly using advanced Unmanned Air Vehicles (UAVs) to perform critical missions. UAVs have grown in capability, while the Navy's underlying Command and Control structure has changed little to take advantage of the advances in technology. While UAVs are rapidly developing the potential to be effective combat tools, learning how to utilize this potential in an integrated Command and Control environment is hampered by a lack of UAV connectivity.

This thesis develops a methodology for using UAV telemetry data packets to inject tracks of the UAV into a Command and Control system such as the Global Command and Control System (GCCS), and provide near-real-time imagery delivery from the UAV to tactical end users via a network such as the Secret Internet Protocol Router Network (SIPRNET). Focus is on the development of a proof-of-concept system utilizing the Naval Postgraduate School's Systems Technology Battle Lab (STBL) and the Center for Interdisciplinary Remotely-Piloted Aircraft Studies (CIRPAS) Altus UAV. Through the developing of this system, the Altus UAV can serve as a research tool for further development of Command and Control doctrine for operational UAVs.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. OVERVIEW

This thesis examines the technical aspects of implementing a UAV track injection and imagery presentation system (UTI^{2}PS). The technology of UAVs is developing quicker than the underlying "intelligence machine" and in fact, is outpacing the advancement of C4I systems. While the bandwidth capacity and technical capability of C4I systems is increasing, the connectivity necessary for UAVs has yet to be incorporated. This thesis provides a technical demonstration of how UAVs can provide positional and imagery data to the end user via current command and control systems such as the Global Command and Control System (GCCS), and communication means such as the Secret Internet Protocol Router Network (SIPRNET). Common off the shelf (COTS) hardware and software as well as open standards are utilized where possible.

B. PURPOSE OF RESEARCH

The purpose of this thesis is to develop a low cost, COTS, near-real-time methodology for getting telemetry and imagery from UAVs to tactical end users. An architecture will be identified which will provide connectivity from a UAV through the Ground Control Station controlling it, to end tactical users via the Global Command and Control System and SIPRNET. High levels of availability, scalability, compatibility, and flexibility will be achieved by using a TCP/IP network-based architecture to as great an extent as possible. This thesis will also demonstrate the technical feasibility of such a proposed architecture by development of a proof-of-concept implementation.

C. SCOPE OF RESEARCH

This research presents the requirements, architectural design, and a proof-of-concept implementation of a prototype system to inject UAV telemetry and imagery information into a command and control system. A proof-of-concept prototype implementation illustrates the conceptual as well as technical viability of the concepts

presented within this thesis. This proof-of-concept demonstration sets the stage for further refinement of this technology, and provides a valuable research tool for the evaluation of UAV Command and Control techniques that will potentially become future doctrine.

The following research questions are addressed:

1. What are the requirements for a UAV track injection and imagery presentation system (UTI^{2}PS)?
2. What is the appropriate architecture for such a system?
3. How can telemetry data be extracted from the UAV Ground Control Station RS-422 port?
4. How can imagery data be captured and encoded at the Ground Control Station?
5. How can telemetry and imagery data be transmitted to the NPS Systems Technology Battle Lab (STBL) or other SIPRNET insertion points from remote locations?
6. How can ANSI/IEEE standard 754 telemetry data packets be converted into Over The Horizon Targeting Gold (OTG) format for injection into Command and Control Systems?
7. How can OTG messages be processed to create a track within the Global Command and Control System?
8. How can imagery, both motion and still, be transferred, stored, and displayed over a TCP/IP network such as the Secret Internet Protocol Router Network (SIPRNET)?

These research questions are answered within this thesis through the development of a proof-of-concept demonstration.

D. THESIS ORGANIZIATION

This thesis is organized as follows: Chapter II further defines the problem statement. Chapter III describes the system Architecture starting at a high level. Chapter IV then provides a detailed discussion of the telemetry methodology as Chapter V does for the imagery methodology. Chapter VI describes the implementation details of the proof-of-concept demonstration and the results obtained. Finally, Chapter VII presents research conclusions and recommendations for further work. Appendices are included

which contain program code developed as part of the proof-of-concept implementation deemed significant.

II. PROBLEM STATEMENT

A. INTRODUCTION

This chapter outlines a set of UAV command and control challenges addressed in this thesis, and the associated limitations in current UAV telemetry and imagery dissemination that must be overcome in order to solve them. It then introduces the focus for the system that will be developed within this thesis to demonstrate means to overcome the data dissemination limitations.

B. COMMAND AND CONTROL

The ability to make timely decisions is a key objective of the command and control process. As our knowledge about a situation increases, so does the ability to make sound decisions. UAVs can provide critical components of the overall tactical picture. However, in order to maximize the benefits UAVs can provide, new command and control structures must be developed, which will require the application of new technology for the dissemination of telemetry and imagery from UAVs.

> Current UAV systems were born largely from advanced technology demonstration projects. They were designed more to validate and test the technology than to operate in standardized command and control environments. While they proved successful operationally, as demonstrated by the Pioneer during the Gulf War and the Predator in Bosnia, they were tasked in a very narrow capacity with dedicated command and control, limited image dissemination, and restricted inter-connectivity.
>
> In the Navy, operational UAV doctrine is based almost entirely on the Pioneer system. The Pioneer UAV has a narrow mission and a corresponding limited operational doctrine. This doctrine does not translate to the increased capabilities and expanded scope the Navy will enjoy with VTUAVs. Furthermore, the Navy must expand its doctrine to encompass HAE UAVs, since these will feed data into naval C4I systems.

> The Navy's IT-21 and JV 2010 policy statements also indicate a paradigm shift towards increasing inter-connectivity and the concept of Network-centric Warfare. In this vision, data from all sensors is available across the network for utilization by war fighters that can parlay a narrow requirement for information into a tactical advantage. The growth in intelligence gathered by UAVs underscores the requirement to integrate them into the C4I environment.
>
> The existing UAVs are all independent systems. They do not inter-connect easily with command and control systems, which isolates them from the lower echelons that need the information. The information is strictly in one form: processed data that is derived from the sensor data. This information must be filtered through a rigid dissemination process that limits the inherent flexibility of UAVs. Imagery to the tactical warfighter and real-time tracks will be important for modern UAVs." [Ref 1: pp. 23-24]

The operational objective is to get UAV positional information and imagery to the war fighter. The war fighter is anyone from the troops in the trenches who are fighting the battle, to the commanders at all levels who are planning for the successful outcome of the operation. Thus there is a need to provide real time information to enhance the war fighters situational awareness.

1. Track Injection

The capability to directly interface with the Global Command and Control System (GCCS), therefore allowing full awareness of a UAV's position (and UAV generated tracks) through track symbology on the Common Operating Picture (COP), will enable track de-confliction and allow commanders to assess utilization and provide direction of UAVs. Currently capabilities do not exist to detect and track a UAV, and to hold the UAV track data beyond organic sensors (OTH), without relying on a commander's organic assets such as radar.

> According to HAE UAV CONOPS, the battle group does not have JMCIS track information on the HAE UAVs to de-conflict them from radar tracks. The HAE UAV CONOPS does not address the dissemination of real-time air vehicle track information from the ground control element to the fleet. Therefore, the battle group must assume that any airborne track is hostile if it does not hold a JMCIS track to merge it with. [Ref 1: p. 28]

Only with real-time track information will commanders be able to monitor the position of UAVs, and effectively integrate them into the war fighter's forces in order to properly employ them.

2. Imagery Dissemination

The UAV imagery dissemination pipeline begins at the air vehicle sensor and, for the purposes of this thesis, ends with the end user. Current UAV image dissemination systems are essentially stove-piped systems, with little or no mechanism to deliver imagery to end tactical users in a timely manner.

Currently, high value units (HVUs) may have the capability to receive imagery time-late, but other ships within a battle group probably do not because the capability for broadband connectivity between ships is limited, and will not support direct broadcast reception of live video feeds. Thus currently a destroyer firing on a target can not use UAV imagery directly to make fire corrections.

However, raw imagery from a UAV is not enough. Imagery should be processed to include vital telemetry information such as location, orientation, time, date and other kinematics data.

> Raw imagery, even when comprehensive and unquestionably accurate, is of little use to the war fighter. It may be a wonderfully clear photographic image of some location, but without additional, vital information, such as the location of the photo, its orientation to true north, and its time and date of origin, the image might be useless. [Ref 2: p. 21]

C. FOCUS FOR SYSTEM DESIGN

1. Center for Interdisciplinary Remotely-Piloted Aircraft Studies

The Center for Interdisciplinary Remotely-Piloted Aircraft Studies (CIRPAS) is an independent research program, administratively associated with the Naval Postgraduate School (NPS), that provides UAV flight services to the research, development, test and evaluation communities. CIRPAS has hanger, maintenance and administrative spaces at the Marina Municipal Airport (formerly Fort Ord's Frizsche Field), located close to NPS. They also have established an unmanned aircraft base of operations at Camp Roberts, CA, an Army National Guard base located approximately 80 miles south of the Marina facility. The UAV base of operations allows unmanned vehicle operations in restricted airspace over Camp Roberts and nearby Ft. Hunter-Liggett. [Ref 3]

Along with the Pelican, UV-18A Twin Otter, and recently acquired Preditors, CIRPAS operates the Altus UAV (Figure 2.1), which was developed by General Atomics, and is similar to the Predator UAV. The Altus is used to support Atmospheric Radiation Measurement (ARM) for the Department of Energy's Sandia National Labs. Because of an agreement with the DoE, CIRPAS is able to provide the services of the Altus to other users when not supporting the ARM experiment. As a result, the Altus is used as a test bed for various developmental payloads, and serves as a surrogate for operational UAVs. Thus the Altus provides an excellent test bed for the development of a UAV track injection and imagery presentation system.

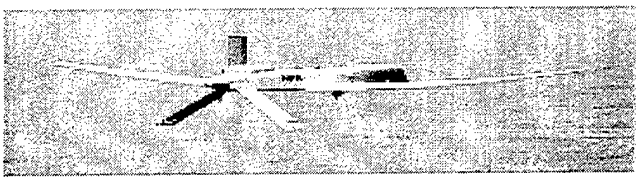

Figure 2.1: Altus UAV

Imagery and kinematic data from the Altus UAV is currently down-linked via C-Band line-of-sight (LOS) to a General Atomics developed Ground Control Station (GCS) which provides aircraft control functions (Figure 2.2). The GCS has redundant pilot / payload operating stations and is housed in a rugged 16-ft long trailer (Figure 2.3). At the GCS, imagery and kinematic telemetry are archived onto tape for future analysis and review. The imagery includes video from a forward-facing camera used to remotely pilot the UAV, and can also include video from a payload camera. The kinematic telemetry includes all data necessary to monitor the UAV status, including oil pressure, prop RPM, servo temperature, GPS position, heading, altitude, and airspeed. This telemetry also includes all necessary components to create a track symbol of the UAV for insertion into the Common Operational Picture (COP).

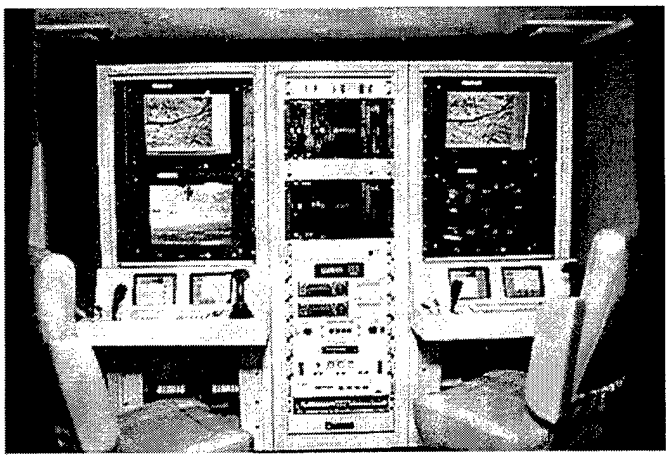

Figure 2.2: Interior of Ground Control Station

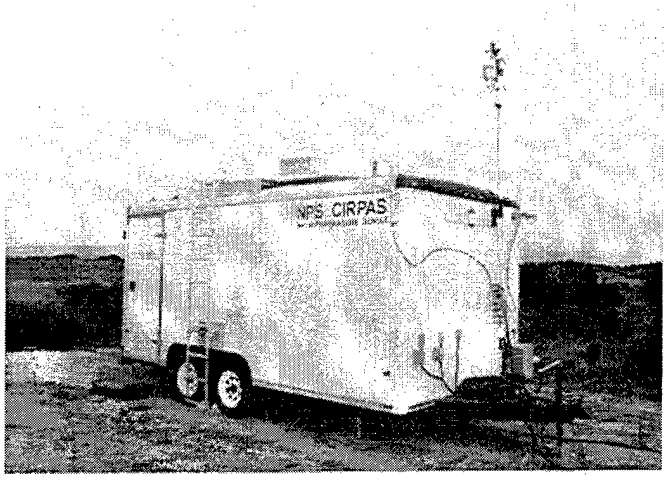

Figure 2.3: Exterior of Ground Control Station

The missing components to enable the creation of tracks of the UAV are links between the Ground Control Station and the existing Global Command and Control System (GCCS) Common Operational Picture (COP) environment. The goal of this research is to develop these links and also demonstrate a means of presenting near-real-time imagery from the Altus UAV to the tactical war fighter via the Command and Control network, or SIPRNET.

2. Developed Methodology

Creating track symbology of the Altus within the GCCS, specifically on the Common Operating Picture (COP), requires several steps. Overall, the telemetry data must be captured at the GCS and converted into a format that can be injected into the GCCS. This will require converting the telemetry from General Atomics proprietary ANSI 754 format into a format that can be read into the GCCS track information database, such as Over The Horizon Targeting Gold (OTG) messages. Next these messages must be transmitted to a location where they can be injected into the GCCS. Finally, these messages will yield the track symbology for the Altus UAV displayed on

the COP, and although not part of this thesis effort, tracks detected by the UAV since that kinematics data is available from the UAV telemetry as well.

Providing near-real-time Altus imagery requires first capturing the live video from the GCS, and then converting it into a standardized digital format. The digitized imagery then needs to be transmitted to a server for archival and subsequent presentation to clients.

D. SUMMARY

The thesis examines the technical aspects of UAV telemetry and imagery dissemination necessary to enable the command and control mechanisms required to effectively leverage the additional capabilities presented by the proliferation of UAVs.

The primary goal of this research is to develop a proof-of-concept implementation of a system that will inject UAV telemetry information into the Global Command and Control System (GCCS) Common Operational Picture (COP), providing real-time UAV positional information to tactical war fighters. An additional goal is to develop a system that will provide imagery from UAVs to tactical war fighters in near-real-time. The CIRPAS Altus UAV will be used as a test-bed for the development of such a system, and can subsequently be used as a research tool for the evaluation of evolving UAV Command and Control doctrine.

III. SYSTEMS ARCHITECTURE

A. INTRODUCTION

This chapter presents the UAV Track Injection and Imagery Presentation System (UTI^{2}PS) architectural model based on the design goals presented in the prior chapter. The goal is to design a system that will provide a subset of the telemetry and imagery data from the UAV to the tactical end user in near real time.

B. THE BIG PICTURE

The function that UTI^{2}PS must perform is to act as an interface between the UAV and tactical end user, providing telemetry data to indicate where the UAV is currently located, and imagery data directed from the UAV payload camera. It is desirable to make this interface between the UAV sensor and tactical war fighter as simple as possible, thus shortening the sensor-to-shooter loop. Ideally such a system would consist of the UAV down- linking positional information and imagery data via a wireless network directly to the tactical end user as shown in Figure 3.1. Due to limitations in current UAV technology, this is not possible, so alternative means must be identified.

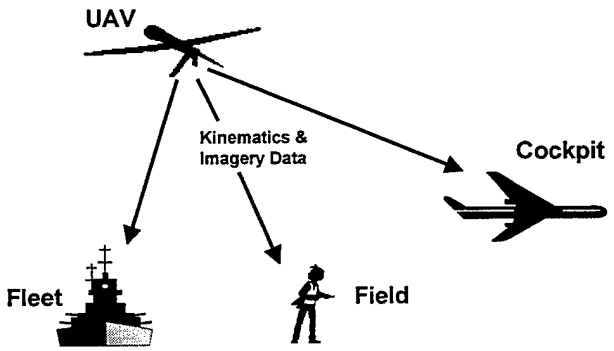

Figure 3.1: Ideal UAV Data Dissemination

Since it is not possible to retrieve data directly from a UAV such as the Altus, which utilizes C-Band LOS communications to a Ground Control Station (GCS), the UTI²PS system must extract the necessary telemetry and imagery data from this intermediate data collection source before it is provided to tactical end users via the C2 net or SIPRNET. This is illustrated in Figure 3.2 below:

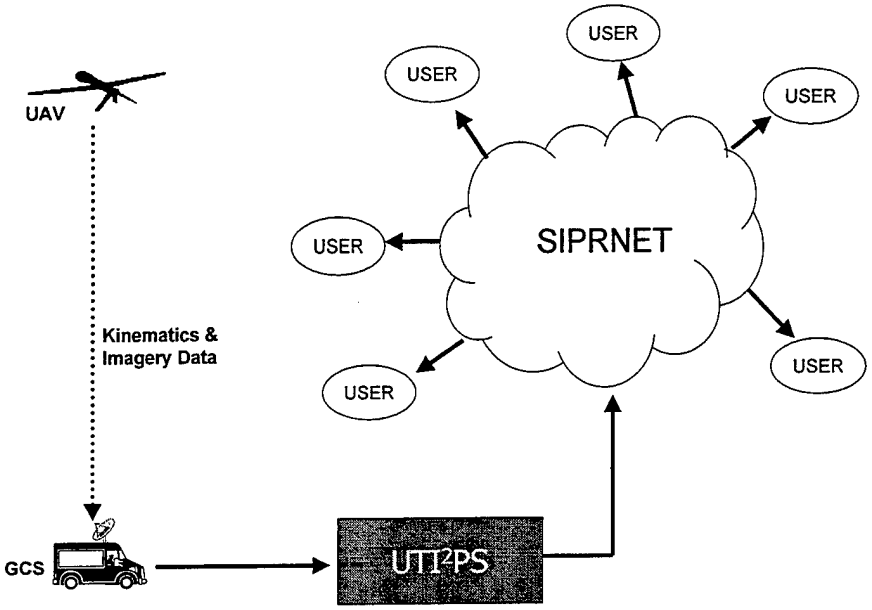

Figure 3.2: UAV Data Dissemination Using UTI²PS

C. UTI²PS ARCHITECTURE

UAV telemetry data and video imagery is fed into the UTI²PS system from the GCS. The telemetry data is converted into OTG messages, which are the primary message format for Tactical Data Processor (TDP) information exchange and fully supported by the Global Command and Control System (GCCS).

Imagery from the GCS is feed into the UTI²PS system and split into two separate feeds to allow for the encoding of a live video stream, as well as the capturing of still digital images. It is desired to provide both streaming video and still imagery because this will best accommodate the perceived needs of the end user, while allowing the users

to manage bandwidth requirements. Providing high-resolution full-motion streaming video requires high bandwidth rates, which are currently not available to most tactical end users. Thus, it was decided that moderate resolution streaming video would be sufficient for end users if a means of providing higher-resolution still imagery for targets of interest could still be provided via an alternative means. Since moderate resolution streaming video, with occasional still digital images of high quality, can be provided using less bandwidth than would be required for high-quality full-motion video, this is a logical choice. If bandwidth to the end user is not available for providing even moderate quality streaming video, or it is deemed unnecessary, only the low bandwidth still imagery need be provided.

When still images are captured, the latest telemetry data indicating time and location of fix is also captured and saved to a database along with the still image. In addition, a kinematics-based OTG message is generated that also specifies the location of the image. Still images can then be retrieved from this database as desired by time, location, or track number.

Near real-time imagery is provided to end tactical users by a steaming media server via the SIPRNET or via the Global Broadcast System (GBS). Still imagery is retrieved from a database repository and provided to end-users over the SIPRNET from a web server. Positional data of the UAV is provided by a symbol on the Global Command and Control System (GCCS) Common Operating Picture (COP), which is generated by injecting OTG messages into a GCCS gateway, and broadcasting to clients over the SIPRNET. Figure 3.3 illustrates the flow of imagery and data from the UAV to the end user by means of the UTI^{2}PS Architecture.

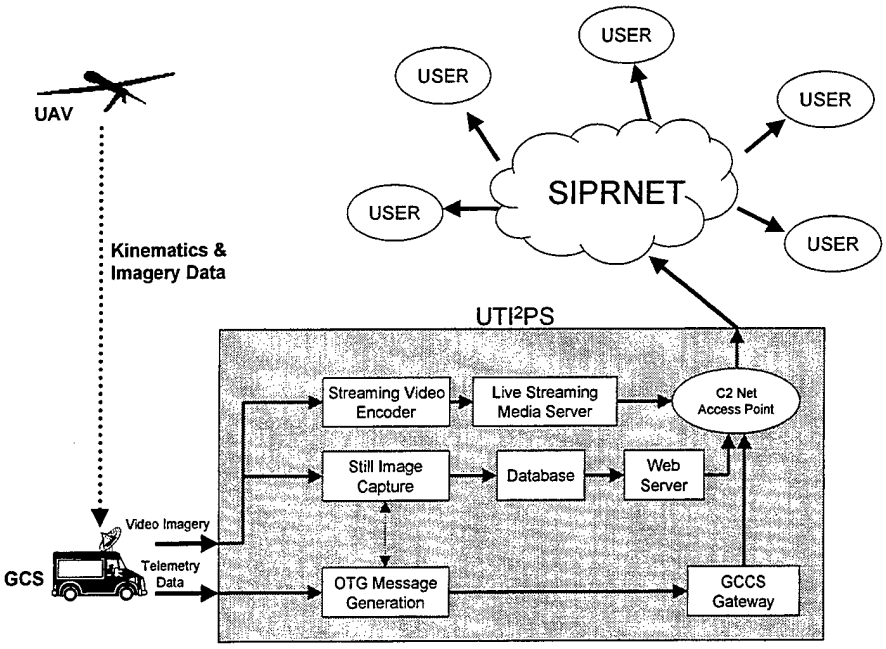

Figure 3.3: UTI²PS Architecture

D. PROOF-OF-CONCEPT ARCHITECTURE

For purposes of this thesis, a proof-of-concept implementation of UTI²PS was designed specifically to provide connectivity between the Altus UAV and the GCCS COP via the Systems Technology Battle Lab (STBL) at Naval Postgraduate School (NPS). Figure 3.4 illustrates the flow of data from the UAV to the GCS via C-Band LOS, then to the NPS STBL on a modem connection over POTS or ISDN, and finally out to end users via the SIPRNET.

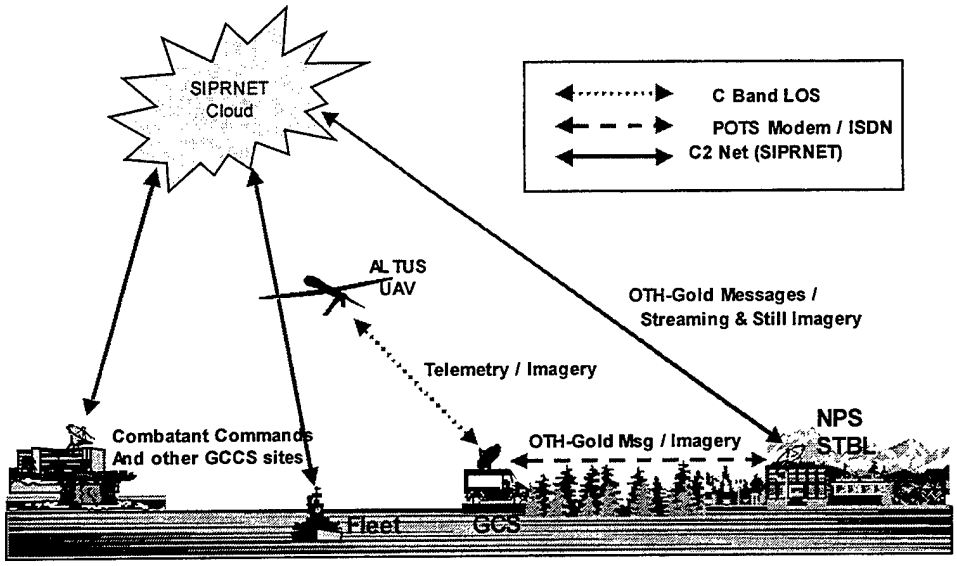

Figure 3.4: UTI²PS Proof-of-Concept Architecture

Figure 3.5 provides a more detailed look at the architecture for implementation of the proof-of-concept. From the GCS, digital telemetry data is transferred into a computer called the GCS Processor via an RS-422 serial interface. Two analog NTSC video feeds containing analog NTSC imagery from the UAV payload camera are also downloaded to the GCS Processor via the GCS. The GCS Processor converts the digital telemetry data into OTG messages, encodes one of the analog video feeds into a digital video stream, and allows for digital capture of still images from the other analog video feed.

Data containing the digital imagery and OTG messages is then transferred from the GCS site to the STBL using TCP/IP and Remote Access Server over a POTS or ISDN modem connection. At the STBL lab, a Server stores still images in a database and provides them to clients as requested via a web server. This server also serves up the digitized video stream to clients, and forwards the OTG Messages to a GCCS gateway via an RS-232 serial interface. Clients within the STBL, or on the SIPRNET, will then receive a track of the UAV on the GCCS COP, and be able to access imagery from the UAV via a web browser and streaming media player.

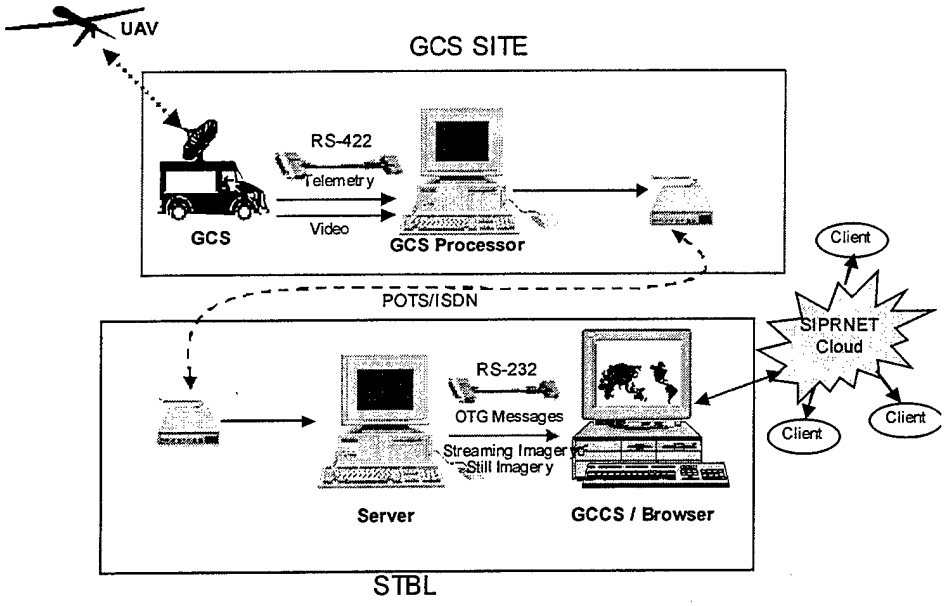

Figure 3.5: Detailed UTI²PS Proof-of-Concept Architecture

E. SUMMARY

The proof-of-concept UTI²PS architecture utilizes common interfaces and open standards where possible, but still requires connection to proprietary non-networked systems via serial means. A network-centric system design where the UAV telemetry and imagery sensors could be directly linked to the GCCS command and control network would be more desirable due to the elimination of potential bottlenecks and single points of failure, but unfortunately this is not currently possible. The next two chapters provide more detail on the telemetry and imagery methodologies utilized in the proof-of-concept implementation.

IV. TELEMETRY METHODOLOGY

A. INTRODUCTION

The previous chapter discussed an overview of the system architecture. This chapter describes the methodology used in developing a system for capture, transmission, and presentation of Altus UAV telemetry data on the GCCS COP. This will enable GCCS users to view the UAV track when over the horizon (OTH). Previously the UAV was tracked using organic assets such as surface to air radar.

B. OVERVIEW

The goal of our digital telemetry research is to design a system that will convert a binary telemetry data packet received for the UAV into a formatted message suitable for injection into the GCCS. A diagram of the overall system architecture is illustrated below in Figure 4.1.

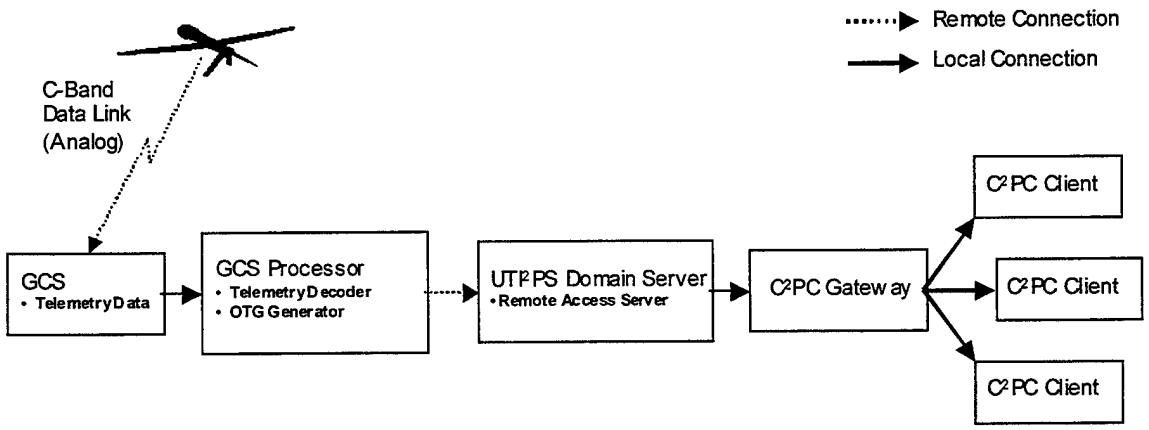

Figure 4.1: Basic UTIP²S Telemetry Block Diagram

The Altus UAV sends digital telemetry data to the Ground Control Station (GCS) using a line of sight C-band communications channel. At the GCS the data is processed for presentation to control and monitor the flight performance status of the UAV. Within the GCS there is the capability to feed a subset of the telemetry data to an external serial port. It is from this port that kinematic data such as latitude, longitude, altitude, course, and speed would be extracted.[1]

The telemetry data is fed to a processor co-located with the GCS where the binary data is converted from ANSI/IEEE-754 (single precision floating point format) to a formatted OTG message. This message is sent to a C^2PC Gateway via a remote access server using TCP/IP. The C^2PC Gateway is part of the Command and Control PC (C^2PC) suite, developed by Inter-National Research Institute (INRI), for Windows NT and is responsible for processing incoming OTG messages and updating the GCCS track information database. Clients that receive the appropriate GCCS feed will receive all messages that originate from the GCS.

C. OVER-THE-HORIZON TARGETING GOLD (OTG)

> The Operational Specification for Over-The-Horizon Targeting GOLD...provides a standardized method for transmitting selected data between OTH-T systems and OTH-T support systems.... It is the primary message format for Tactical Data Processor (TDP) to TDP information exchange on the Officer in Tactical Command Information Exchange System (OTCIXS) and Tactical Data Information Exchange System (TADIXS). It is designed to be easily man readable for the non-TDP user. [Ref 4: p 1-1]

The OTG format is based on a set of message text formats that is fully described in the Operational Specification for Over-The-Horizon Targeting GOLD (OS-OTG) of 1 August 1997. Each line in the OTG message is limited to 69 characters in length and formatting rules must be carefully observed for the message to be processed by the C^2PC

---

[1] In the current software build, however, this data feed is disabled. The next software build, scheduled for April 1999, will restore this feature.

Gateway. Annex 3B (Contact Report) of OS-OTG is of particular interest in generating track data from the telemetry data packet and is discussed here. The selection of the OTG specification provides a standard message format for communicating proprietary telemetry data to GCCS users.

1. Contact Report

"The Contact Report message is used to exchange processed contact data and track management information between systems. It contains the identity, location, and movement of surface, subsurface, land, and air contacts." [Ref 4: p 3B-1] For our purposes, the only sets required for telemetry track injection into GCCS are MSGID; CTC and POS, which together comprise the contact segment; and ENDAT. Figure 4.2 shows a sample OTG contact report.

```
[MESSAGE HEADER]

MSGID/NCTSI/GOLD/0001/JUN
CTC/T7062/WICHITA-ROANOKE////////09
POS/130500Z9/JUN/8500S3/00000W0
ENDAT

[END OF MESSAGE SEQUENCE]
```

Figure 4.2: Sample OTG Contact Report

a. MSGID

The MSGID set is the first formatted set of the OTG message and marks the beginning of formatted sets. The MSGID set uniquely identifies the message by originator of the message, type of message, and serial number.

b. Contact Segment

The contact segment contains two sets: CTC and POS. The POS set must immediately follow the CTC set. Multiple contact segments are permitted with the OTG message but for our purposes only one segment is required. The CTC set contains description data and the POS set contains time and position data of the contact.

c. ENDAT

The ENDAT set specifies the end of the OTG message and contains no data.

D. TELEMETERY SOURCE AND STRUCTURE

Telemetry data sent from the UAV for output through the GCS external serial port is structured in a binary data packet and framed with data link escape codes. Occurrences of data link escape codes within the data must be repeated, a method known as stuffing. These packets are sent at a rate of 10 Hz. The port is configured as follows: 19,200 baud, 8 data bits, 1 stop bit, no parity, and no handshaking. [Ref 6]

1. Packet Structure

Telemetry data from the GCS is structured in a 65-byte packet and is shown in Figure 4.3. The start of each packet is marked with the two-byte sequence <DLE><ID> while the end is marked with the sequence <DLE><ETX>. <DLE> is defined as hexadecimal 10 (ASCII code for Data Link Escape) and <ETX> is defined as hexadecimal 03 (ASCII code for End of Text). <ID> can be any character other than <DLE> or <ETX>. Packet data shall be any eight-bit value with the requirement that all data bytes equal to <DLE> must be sent twice. The receiving device must compress all occurrences of two <DLE> characters into one <DLE> data byte. [Ref 6]

Start		Telemetry Data	End	
<DLE>	<id>	61 Bytes (without stuffing)	<DLE>	<ETX>

Figure 4.3: Telemetry Data Packet

2. Data Structure

Telemetry data fields are described in Figure 4.4. The fields of interest are latitude, longitude, altitude, heading, and speed, as these are used to build the OTG message. Time of fix is not used and will be addressed as an area of future research.

Data Byte	Item	Source	Units
0-3	Latitude	Litton	Radians (+ North)
4-7	Longitude	Litton	Radians (+ East)
8-11	Altitude	Litton	Meters, MSL
12-15	Time of Fix	Litton	GPS Time
16-19	Airplane Pitch	Litton	Radians (+ Up)
20-23	Airplane Roll	Litton	Radians (+ Right wing down)
24-27	Airplane Heading	Litton	Radians
28-31	Latitude	General Atomics	Radians (+ North)
32-25	Longitude	General Atomics	Radians (+ East)
36-29	Altitude	General Atomics	Meters, MSL
40-43	Time of Fix	General Atomics	GPS Time
44-47	Airplane Pitch	Vertical Gyro	Radians (+ Up)
48-51	Airplane Roll	Vertical Gyro	Radians (+ Right wing down)
52-55	Airplane Heading	Magnetometer	Radians
56-59	True Airspeed		Knots
60	Reserved		

Figure 4.4: Telemetry Record Structure

3. Field Structure

The fields in the telemetry data packet are ANSI/IEEE 754 single precision floating point format. This format is defined as 32 bits long and consists of one sign bit,

an eight bit exponent, and a 23 bit mantissa. The first byte contains the sign bit and first seven bits of the exponent, the second byte contains the least significant bit of the exponent and seven most significant bits of the mantissa, while the remaining two bytes complete the mantissa. This format is shown in Figure 4.5. [Ref 6]

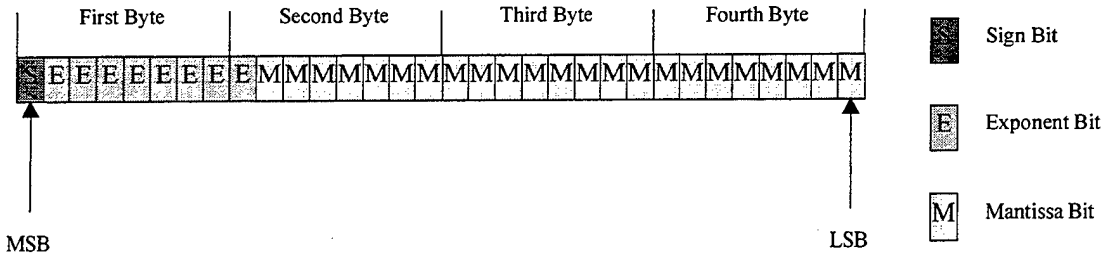

Figure 4.5: ANSI/IEEE 754 Single Precision Floating Point Format

E. DATA CONVERSION

Telemetry data is sent from the GCS to the GCS processor for conversion to a format more suitable for injection into GCCS. Since data is sent from the GCS using RS-422, either an RS-422 port must be installed on the GCS processor or an RS-422 to RS-232 converter must be used. The serial port on the GCS processor must be configured as follows: 19,200 baud, 8 data bits, 1 stop bit, no parity, and no handshaking. The telemetry data packet is parsed and each element is converted from ANSI/IEEE-754 to ASCII text. The resulting text fields can then be used to generate messages for injection into the C^2PC Gateway

The OTG message generation process is shown in the accompanying flowchart (Figure 4.6), and is implemented using Microsoft's Visual C++. Visual C++ was selected because of its compatibility with commercially available ActiveX controls and the requirement to handle a wide variety of data types at the machine level. Appendix A contains the C++ code for converting the telemetry data into OTG messages.

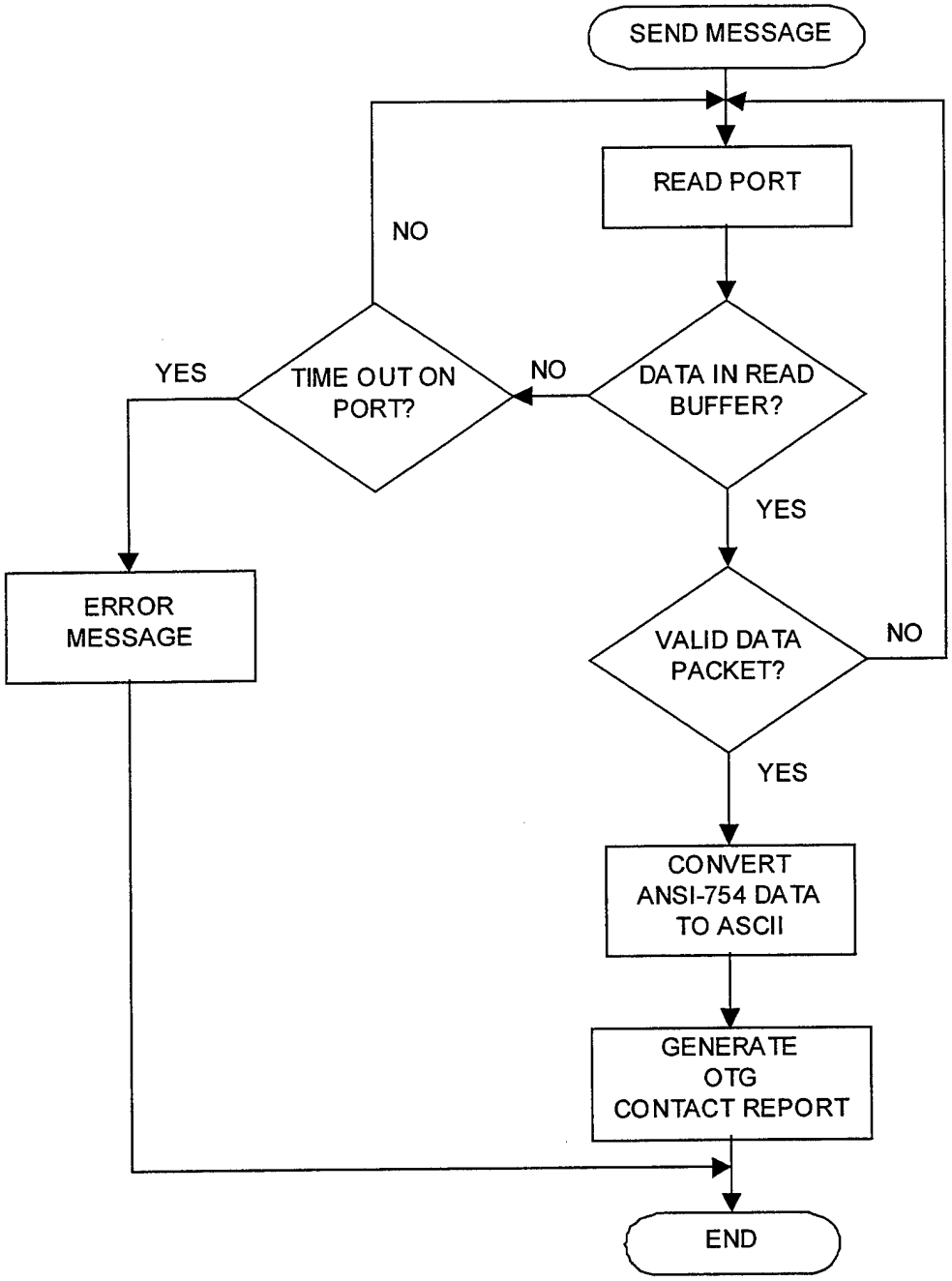

Figure 4.6: OTG Contact Report Generation Process

The conversion process starts when the user presses either the "Send Message" or "Auto Report" button on the user interface (Figure 4.7). The user interface, like the rest of the UTI^{2}PS, was created by the authors. In the case of "Auto Report," a windows timer object is instantiated and every 20 seconds thereafter the process is run. The ASCII representation of the converted data and the OTG message are displayed on the interface. The "Capture Frame" and "Adjust Video" buttons are for capturing imagery and are discussed in Chapter V. See Appendix A for the code used to create the user interface.

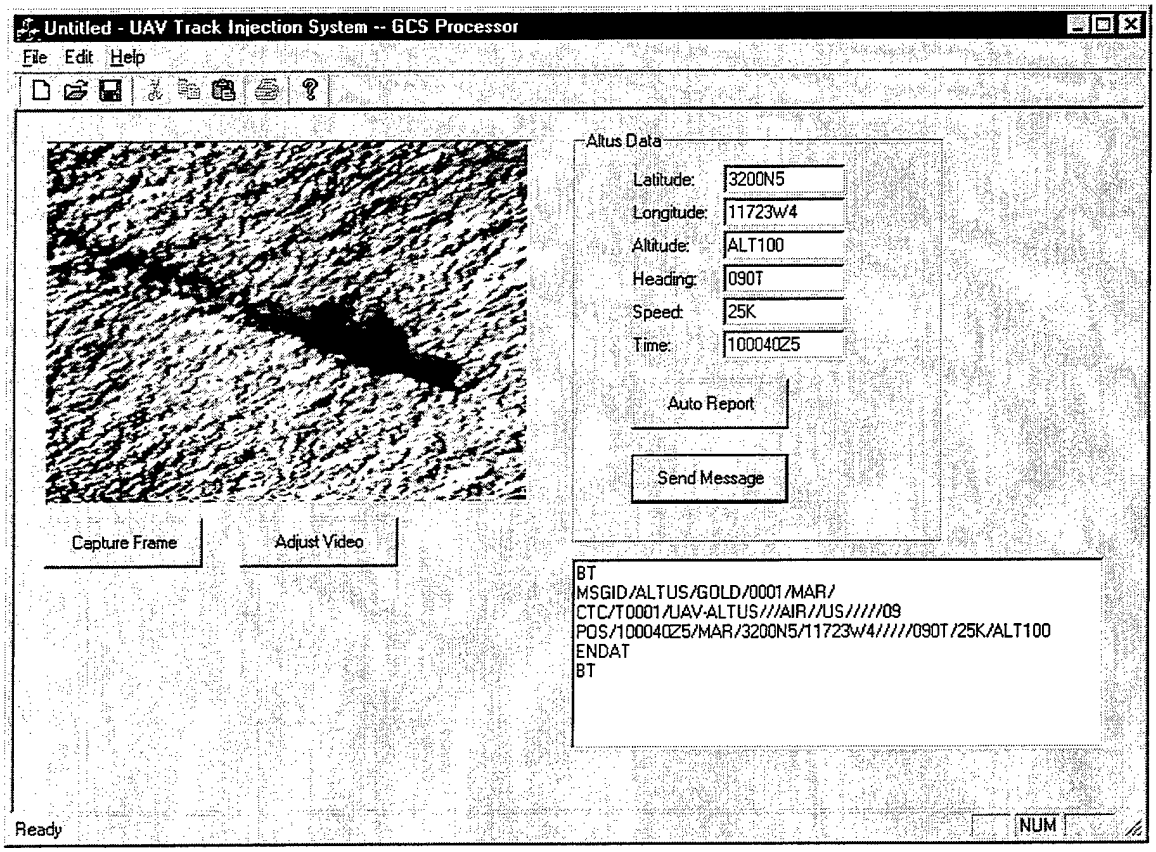

Figure 4.7: UTIPS User Interface

1. Receiving Data

Telemetry data packets are received through an RS-422 serial port installed on the GCS processor. Data received from this port is loaded into a buffer in memory. This buffer contains the data packet sent from the GCS, which must be checked for frame integrity and stuffing of <DLE> data bytes. If a framing error is detected processing stops and the process restarts when a valid frame is detected. If no data is received a time-out error, indicating communications problems, terminates processing.

2. Telemetry Data Conversion

The receive buffer is copied byte-for-byte using a function that strips the data link escape codes and compresses duplicate <DLE> data bytes into another buffer which is defined using a union of data types. By using a union the buffer containing the telemetry data can also be accessed using the structure data type associated with the union. Each field in the structure is converted to an ASCII equivalent text string, which is appropriately formatted for use with an OTG Contact Report and stored in another data structure that is passed to the message generation routine. [Ref 7]

a. Latitude

Latitude data from the GCS is measured in radians, with North expressed as a positive value and South as a negative value. The value must be converted to degrees and minutes and the sign expressed as an "N" or "S". A checksum is required.

b. Longitude

Longitude data from the GCS is measured in radians, with East expressed as a positive value and West as a negative value. The value must be converted to degrees and minutes and the sign expressed as an "E" or "W". A checksum is required.

c. Altitude

Altitude data from the GCS is measured in meters. The value must be converted to feet. A checksum is not required.

d. Heading

Heading data from the GCS is measured in radians. The value must be converted to degrees. A checksum is not required.

e. Air Speed

Altitude data from the GCS is measured in knots. A checksum is not required.

3. Other Data Conversion

The OTG contact report requires additional data in order to be successfully processed. These are reporting unit; message serial number; time and date; track number; class type and name; air, surface, or subsurface.

a. Reporting Unit

Reporting unit is the name of the unit reporting the contact. As our intention is to design a self-reporting system, this is hard-coded as "ALTUS".

b. Message Serial Number

Message serial number is a sequential order, for the calendar month, in which the message was sent. This number is stored as an integer in the GCS processor and is incremented each time a message is sent. The integer value must be converted to a formatted ASCII text string.

c. Time and Date

Time and date is the time and date of the position report. The day of the month and time (in GMT) is extracted and converted to a formatted ASCII text string, which is concatenated with "Z", the time zone indicator for GMT, and a single digit checksum. The three-letter abbreviation for month is also extracted.

d. Track Number

Track number is the local track number of Altus generated tracks. The Altus UAV will always report itself as track "T00001." Other tracks reported by Altus through the imaging sub-system are sequential beginning with "T00002" with an upper limit of "T99999."

e. Class Type and Name

Class type and name is the unit class and unit name of the subject of the contact report. In the case of Altus UAV position reports this is hard-coded as "UAV-ALTUS". For other tracks this is hard-coded as "UNEQUATED-UNKNOWN".

f. Air, Surface, or Subsurface

Air, surface, or subsurface specifies the contact as an air, surface, sub-surface contact. In the case of Altus UAV position reports this is hard-coded as "AIR". For all other tracks this is left blank.

4. Message Generation

We now have the pieces required to generate an OTG message. A message template is initialized from string tables stored as an application resource. The dynamic data content is then loaded into the message template to complete the formatted OTG message.

a. Template

The OTG message template (Figures 4.8 and 4.9) is initialized each time a message is created. This is because the Altus UAV position report differs from the contact report generated by the UTI^{2}PS imagery subsystem.

MSGID/ALTUS/GOLD/<Serial>/<Month>
CTC/T0001/UAV-ALTUS///AIR//US/////09
POS/<Time>/<Month>/<Latitude>/<Longitude>/////<Heading>/<Speed>/<Altitude>
ENDAT

Figure 4.8: Altus UAV Position Report Template

MSGID/ALTUS/GOLD/<Serial>/<Month>
CTC/<Track Nr>/UNEQUATED-UNKOWN/
POS/<Time>/<Month>/<Latitude>/<Longitude>///////
ENDAT

Figure 4.9: Contact Report Template

b. Dynamic Data

The dynamic data is loaded into the message templates by inserting the data strings into their respective field positions. In the case of the imagery contact report the position data is that of the UAV itself rather than that of the image. However, the unique track number forces GCCS to create a new track on the COP. Actual contact position is left as an area for future research and is discussed further in Chapter VII.

5. Conversion Complete

With the generation of the OTG message, telemetry data conversion is complete. To be of use, the message must be routed to an OTH-T system that will provide track information to users with SIPRNET access to the GCCS database.

F. MESSAGE ROUTING

The OTG contact report is wrapped in a TCP/IP packet with the destination address set to a host computer running a GCCS relay process. The relay process unwraps the OTG message and redirects it through a serial port connected to the C^2PC Gateway. The C^2PC Gateway in turn provides track information data to connected GCCS users. [Ref 8: p. 111]

1. TCP/IP Transport

The OTG message is sent from the GCS processor to the GCCS relay using TCP/IP. Therefore the only requirement in sending the message is a network connection that supports TCP/IP which can be anything from a point-to-point dial-up connection to a RAS server to a satellite link.

Sending the OTG message using TCP/IP requires opening a connection to the GCCS relay specifying the host address and port number. The host address can be given as either the fully qualified domain name or the decimal dot address of the host. The port number is 2071. The combination of address and port provides a host socket address to which the message is sent. When the GCCS relay acknowledges the request, the OTG message is sent and the connection is closed. This is done using the CSocket class of the Microsoft Foundation Class for Visual C++ and is completely transparent to the user. [Ref 9]

2. GCCS Relay

A relay is required because the GCCS will only accept OTG messages through a serial communications connection. The GCCS relay listens on port 2071 for OTG messages. When a message is received it is redirected to the serial port without error checking and passed on to the C^2PC Gateway for processing. Because a serial port is used to send data to the C^2PC Gateway it is possible to establish a one-way communications path between the relay host and the C^2PC Gateway. The GCCS relay

displays the OTG message as passes through the host (Figure 4.10). Appendix E contains the code used to generate the GCCS relay.

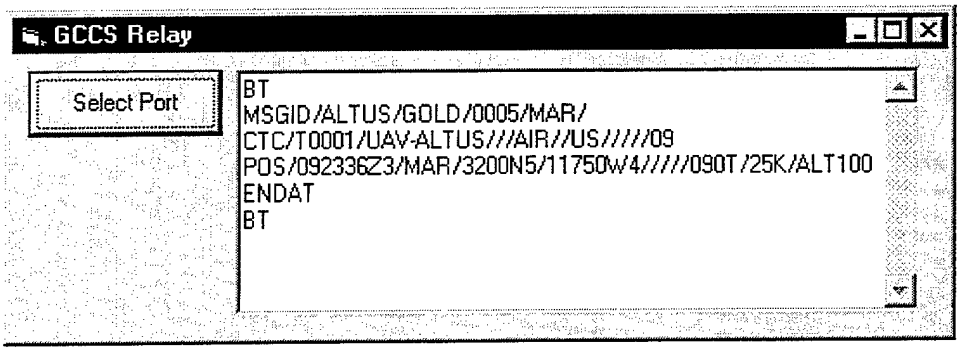

Figure 4.10: GCCS Relay Display

3. C²PC Gateway

The C²PC Gateway receives OTG messages through a serial port. To enable the serial option on the C²PC Gateway GoldDB (Figure 4.11) must be selected as the data source. This is done from the Tools menu in the C²PC Gateway application. [Ref 8: pp. 112-113]

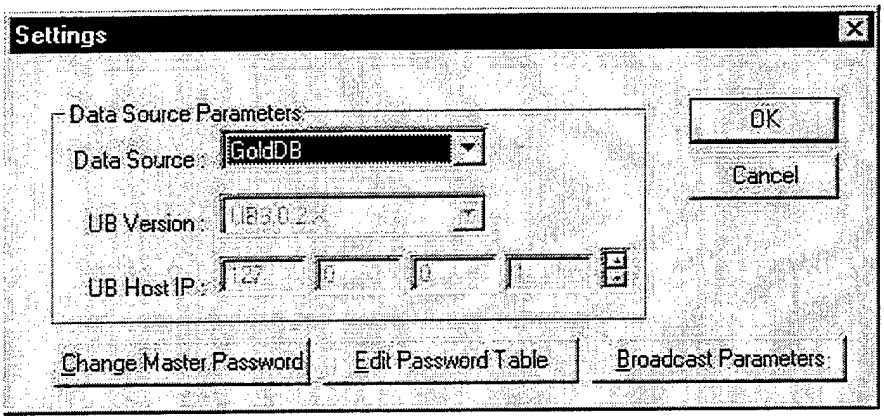

Figure 4.11: C²PC Gateway Settings Dialog

The serial port for the C²PC Gateway is controlled from the C²PC Serial application and must be configured to match the settings of the GCCS relay. Serial must

be running in order for messages to be processed by the C²PC Gateway. OTG messages will be displayed as they are received (Figure 4.12).

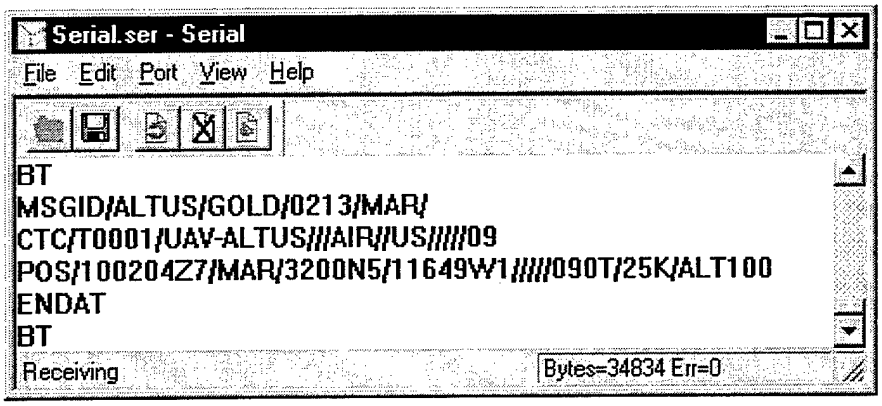

Figure 4.12: C²PC Serial Application

Telemetry processing is complete when the OTG contact report is received and processed by the C²PC Gateway.

G. GCCS COP DISPLAY

When an OTG message is processed by the C²PC Gateway the track information database is updated. The updated position (or new track) will be sent to GCCS clients and the COP refreshed (Figure 4.13).

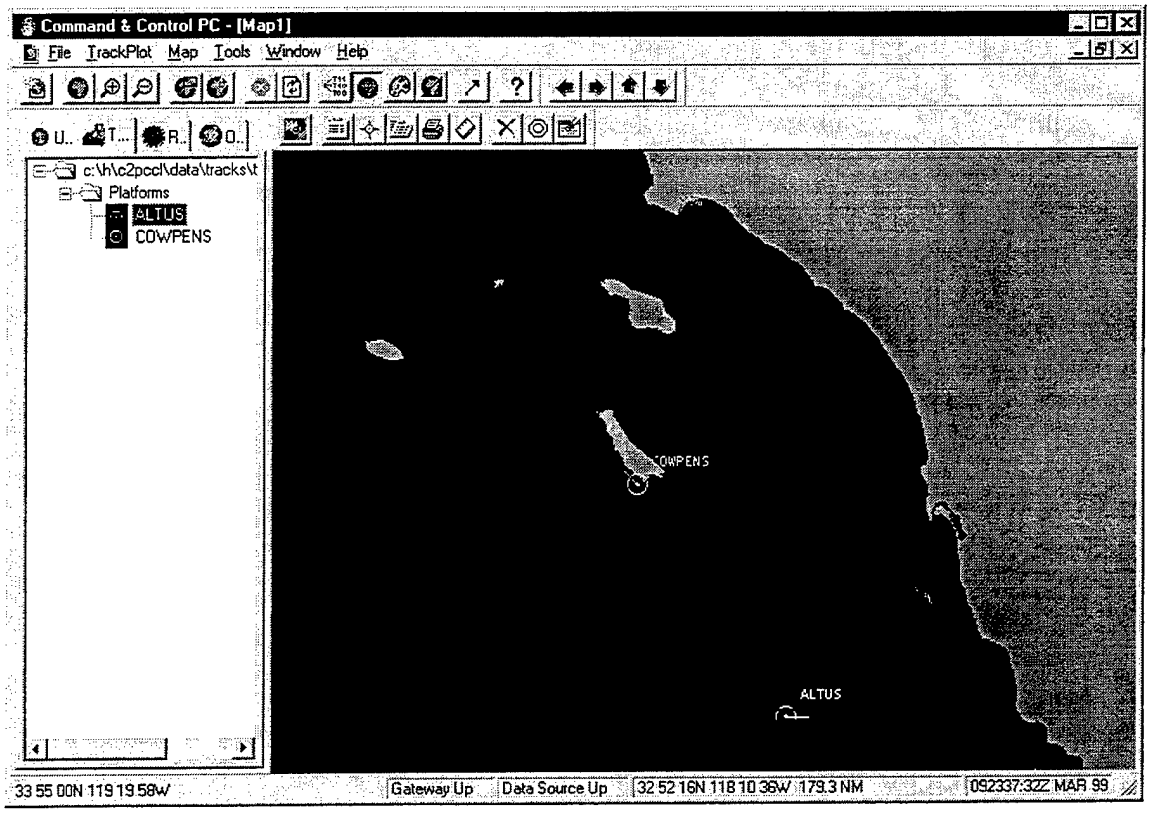

Figure 4.13: GCCS C²PC COP Display

V. IMAGERY METHODOLOGY

A. INTRODUCTION

The previous chapter discussed pivotal telemetry methodology, this chapter describes the methodology used to develop a system for the capture, transmission, and presentation of Altus UAV imagery data. Streaming digital video along with captured images in digital format will be provided to a web server, for viewing over the SIPRNET. In addition to the UAV kinematics track data provided for display on the GCCS COP, the GCCS user will also be able to view near-real-time digital imagery sent by the UAV.

B. OVERVIEW

The goal of this area of research is to design a system using commonly available off the shelf (COTS) hardware and software that will provide for the seamless wide dissemination of near real time imagery from the Altus UAV. A diagram of the overall system architecture is illustrated in Figure 5.1 below.

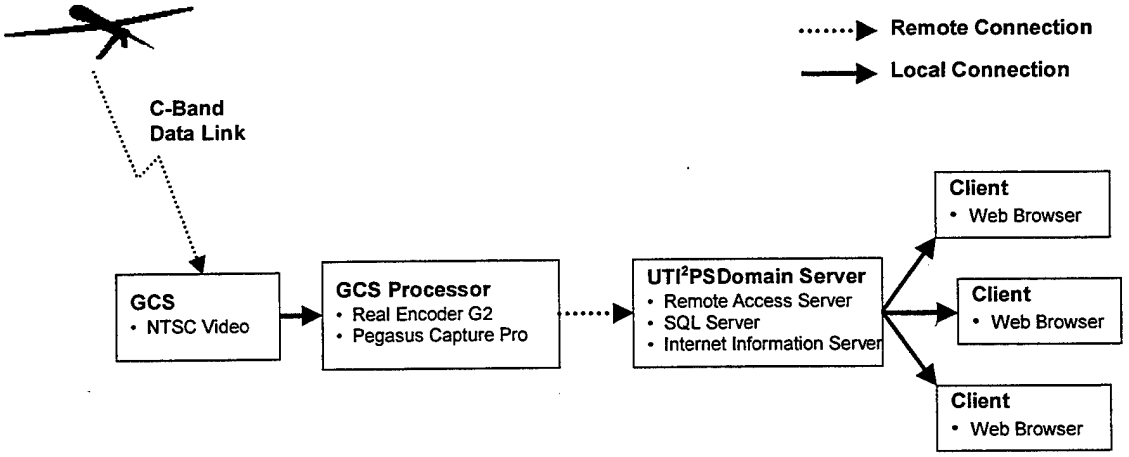

Figure 5.1: Basic UTI²PS Imagery Block Diagram

The Altus UAV transmits imagery from a payload camera to the Ground Control Station (GCS) via RF in NTSC analog format. From the GCS, this imagery is directly fed into a computer workstation located at the GCS site. This computer will be referred to as the GCS Processor. After the imagery is captured and processed by the GCS Processor, it is transmitted to a server at a SIPRNET access site that then inserts live streaming video and repository still imagery to any number of SIPRNET clients via a web browser interface.

The imagery interface between the GCS and remote processor is standard NTSC composite format video. After capture and encoding, digital imagery data is transmitted to the Server in TCP/IP packets over a modem connection. The Server then streams near-real-time imagery over a TCP/IP network to clients, and provides a database repository of still images which clients can remotely access via a web browser as desired.

C. DATA SOURCE

The source of imagery is the Altus UAV payload camera. From the UAV, this imagery is transmitted via C-Band line-of-sight (LOS) to the GCS. This imagery is then provided in standard NTSC composite video format on jack J-11 of the GCS input/output panel. The GCS input/output panel is show in Figure 5.2.

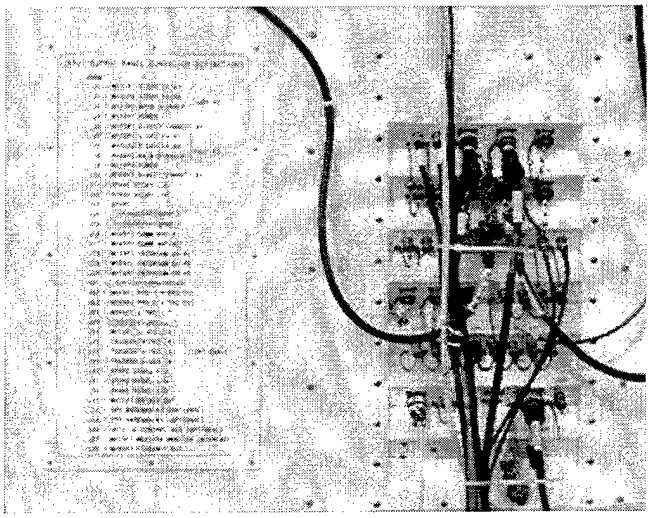

Figure 5.2: GCS Input / Output Panel

This source of imagery contains the raw imagery only, without any telemetry overlays showing position data for the imagery. Imagery with overlays showing telemetry data such as location, orientation to true north, and time & date is available on the CRT displays within the GCS. Providing imagery with these overlays would be possible by tapping off of the RGB inputs going into the CRT displays within the GCS. A tap from these RGB inputs could then be feed into a RGB to NTSC composite video converter and used as the imagery data source rather than the output from jack J-11.

Ideally, imagery data should be provided directly from the GCS, and even the UAV, in digital format such as JPEG. In general architectural engineering terms, the closer the processing is to the data source the better. This would eliminate the necessity to convert the imagery to NTCS, and then digital format as described in the following section. Based on current architecture though, this is not possible, necessitating the methodology used.

D. IMAGERY ACQUISITION REQUIREMENTS

This section describes hardware and software requirements for acquiring imagery from the GCS for distribution on the network.

1. Hardware Requirements

From the GCS, imagery is transferred in NTSC video format to a video capture device located in a computer workstation that digitizes and processes the video, and is thus referred to as the Remote Processor. The minimum recommended hardware requirements based on the software being utilized for imagery acquisition is a Pentium II-266 MHz computer with 64 Mbytes of RAM.

Because it is desired to provide a continuous live imagery stream from the UAV, while simultaneously capturing still imagery, two separate capture devices are utilized. The capture devices utilized are Osprey-100 Peripheral Component Interconnect (PCI) bus digital video capture cards. The Osprey-100 was chosen because of its advertised compatibility with the software being used for acquisition, and because multiple Osprey-

100 video capture cards can be installed in the same computer, preventing a requirement for separate remote processor workstations. Both video capture cards are feed with the same standard NTSC composite video source from the GCS, by splitting the signal and feeding it into each of the Osprey-100 capture boards.

2. Software Requirements

Windows NT Workstation 4.0 is installed as the operating system on the GCS Processor. The software driver that accompanies the Osprey-100 capture cards must be installed. Providing both streaming video and still imagery to warfighters requires additional software.

a. RealProducer

Streaming video requires encoding a standard video signal into a digital stream for transmission on the network. RealProducer G2 from RealNetworks is a commercially available encoder and operates as a stand-alone application. It was selected because of its widespread commercial use, has been determined to deliver the highest quality imagery on the market, and it is free to use for evaluation purposes. [Ref 5]

b. CapturePRO

Capturing still imagery requires capturing a standard video signal and converting it to a digital image. CapturePRO ActiveX from Pegasus Software control is a commercially available programming toolkit for embedding video acquisition in Windows programs. As an ActiveX control it provides the methods and properties that are needed to preview and capture imagery. Pegasus CapturePRO was selected due to the industry-leading reputation of Pegasus Software in the imaging tool arena, and because a free trial version of the ActiveX control for CapturePro was available.

E. STREAMING VIDEO

With RealProducer, providing streaming video content is easily accomplished using the RealMedia Recording Wizard. The operator starts the program, which automatically starts the wizard, and follows the directions.

1. New Session

The first requirement is to select the type of media clip that it is to be created, which in our case is a live broadcast (Figure 5.3).

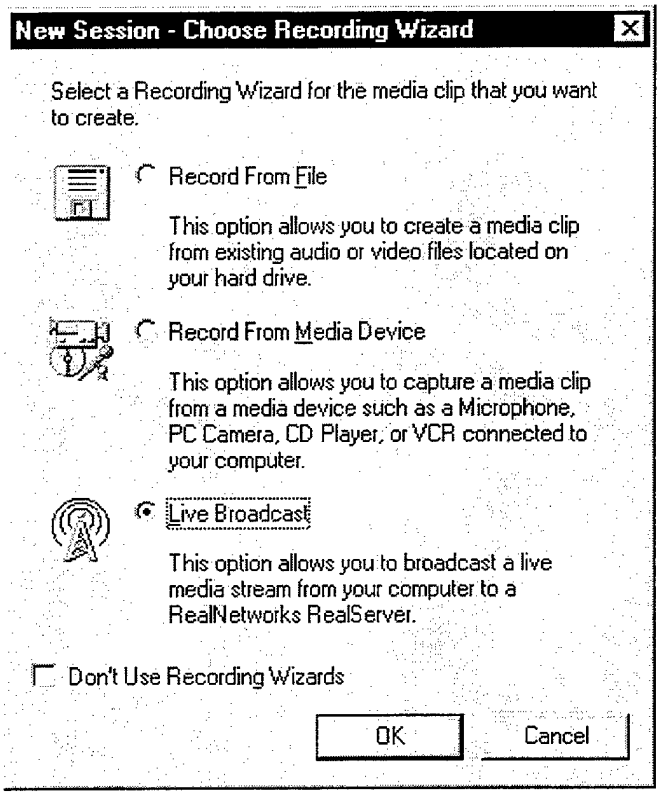

Figure 5.3: Recording Wizard – New Session

2. Input Source

Next, the type of media to capture and source of this media must be set. In our case the type of media is video, and the source is set to one of the Osprey-100 video capture cards (Figure 5.4).

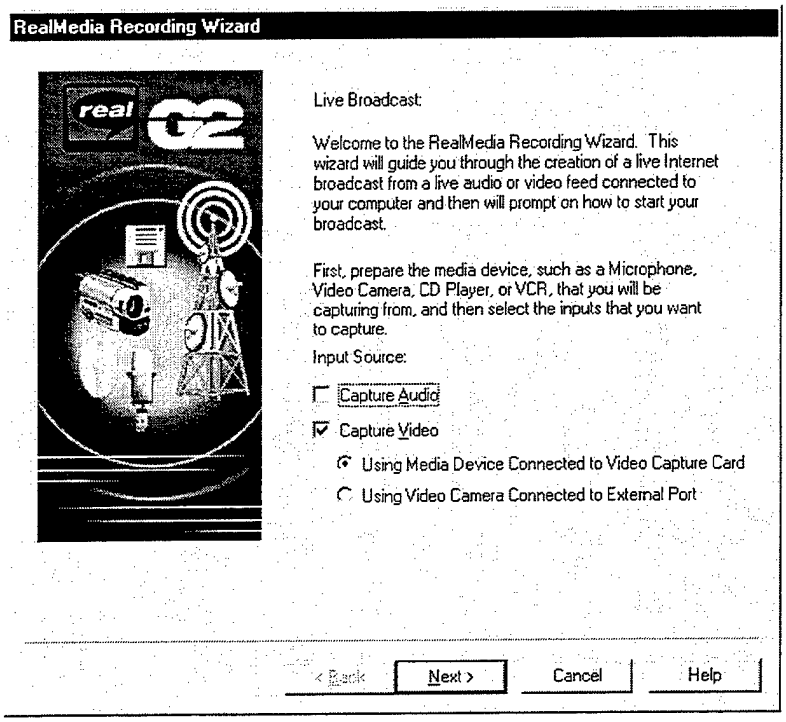

Figure 5.4: Recording Wizard – Input Source

3. Title and Author

A title and author of the broadcast is also set (Figure 5.5).

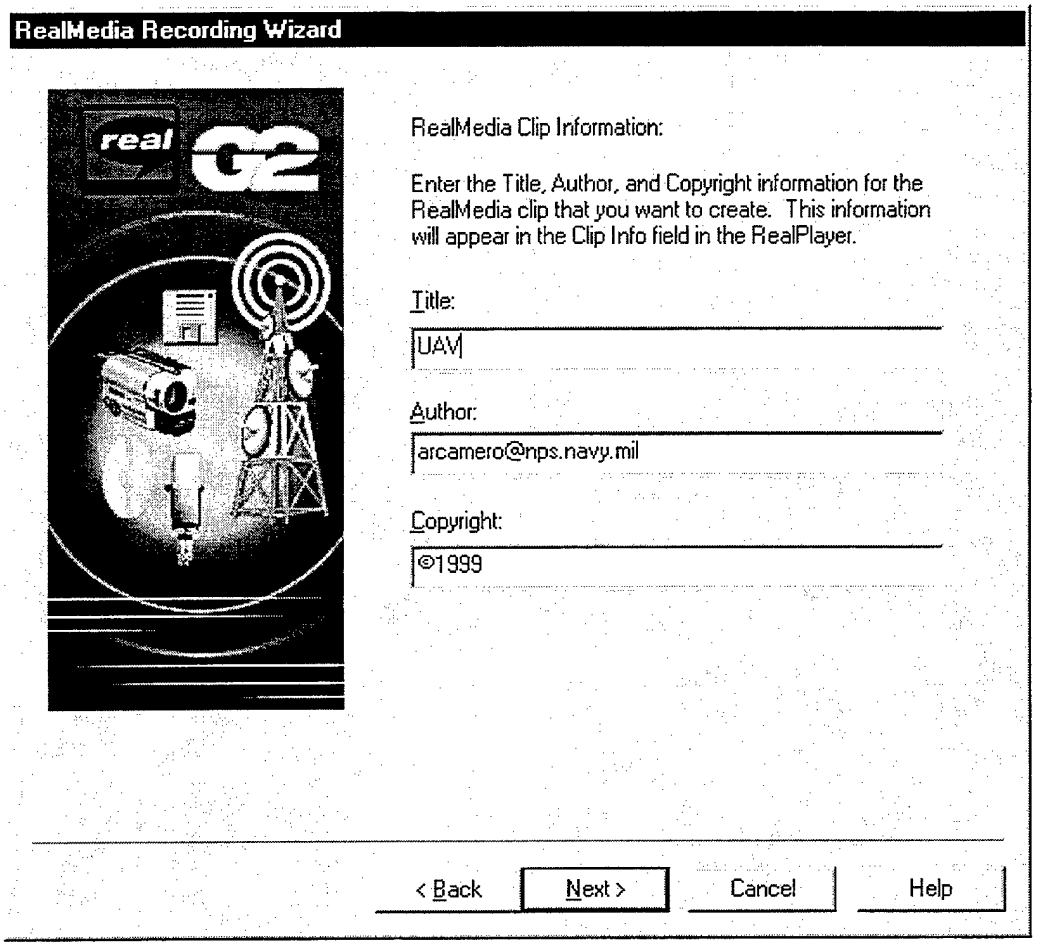

Figure 5.5: Recording Wizard – Title and Author

4. File Type

The file type for the media clip must be selected. The options for file type are SureStream and single rate. SureStream is a scalable format that compresses multiple connection bandwidths into a single stream that dynamically adjusts data flow to changing line conditions. Single rate will provide only a single bandwidth of streaming video from a web server. In our implementation, file type was set to single rate since only one video stream of constant bandwidth was demonstrated (Figure 5.6).

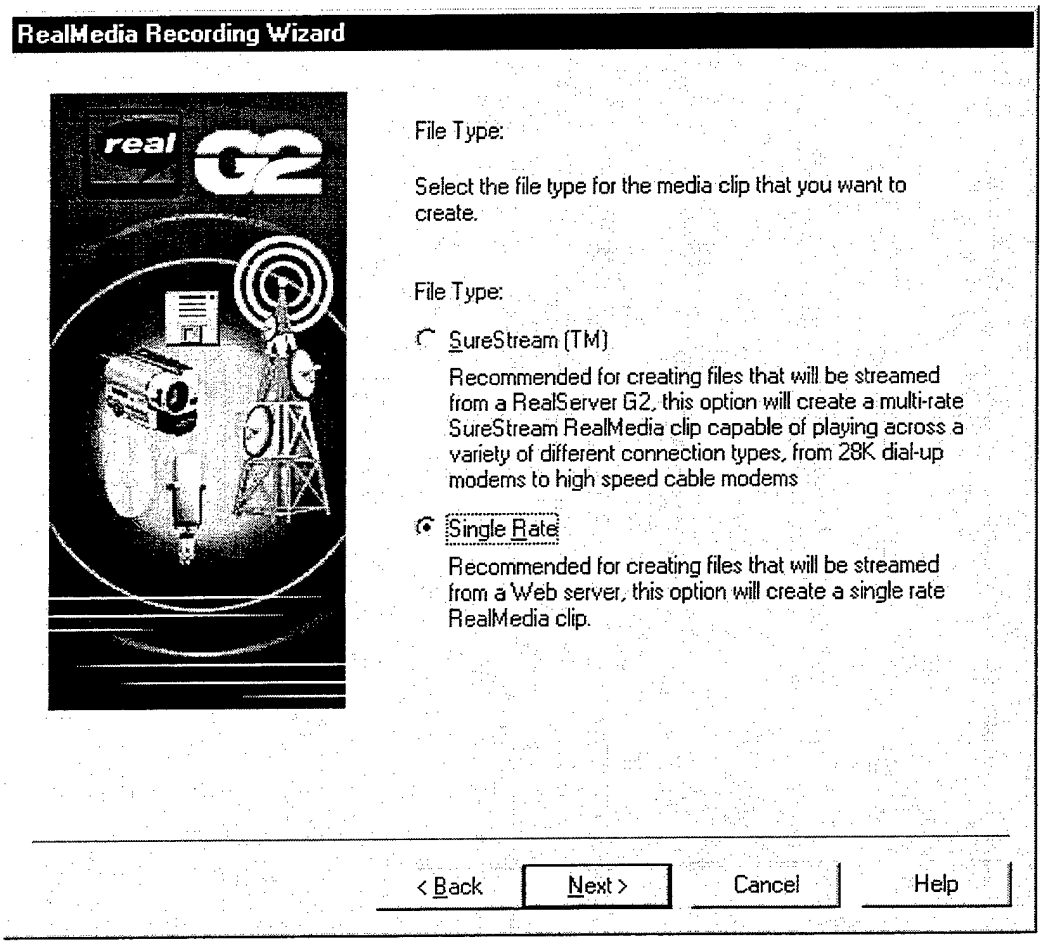

Figure 5.6: Recording Wizard – File Type

5. Target Audience

The target audience data rate setting effects the image quality and frame rate of the digitized video stream, but must be set within available bandwidth limitations. Various target audience data rate settings were tried, with a determination that this must be set at no greater than single rate ISDN to ensure that the maximum tested bandwidth of 115 kbps between the GCS Processor and Server is sufficient (Figure 5.7).

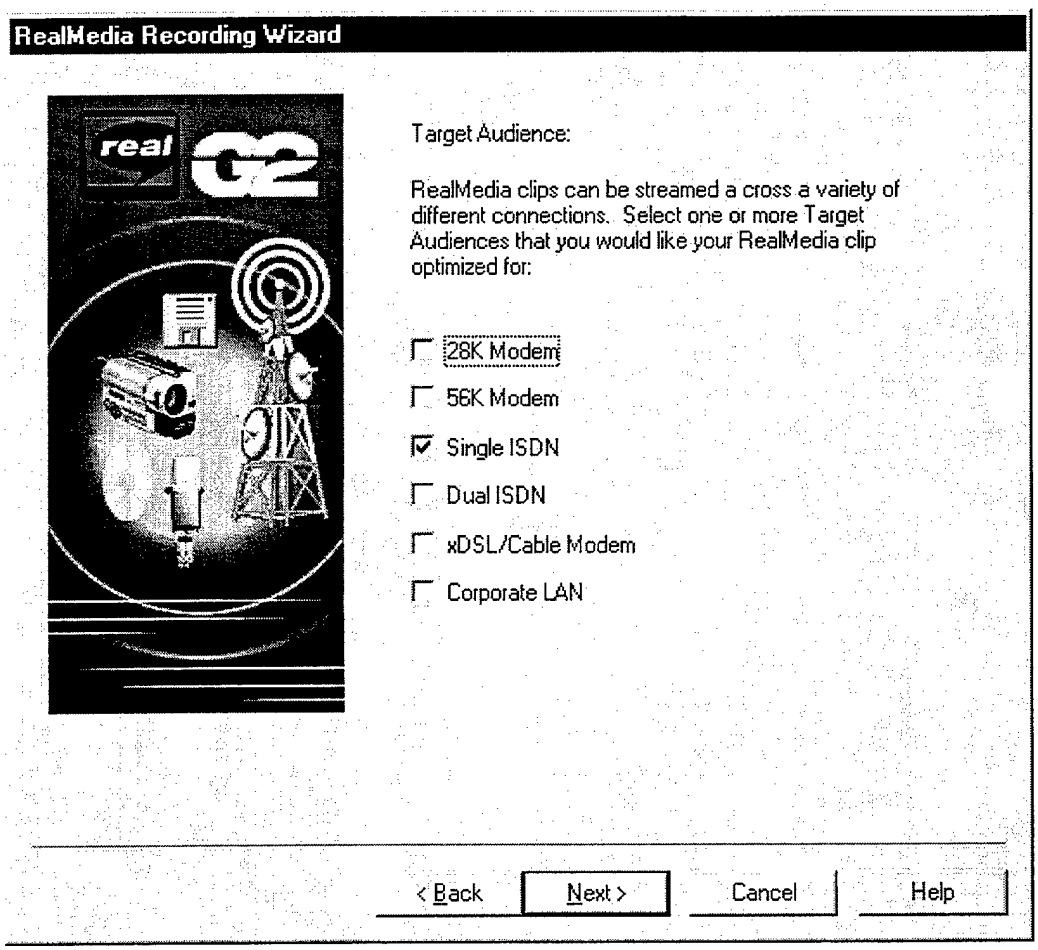

Figure 5.7: Recording Wizard – Target Audience

6. Video Quality

It was also determined that in order to maximize image quality, the video quality setting should be set to slide show (Figure 5.8).

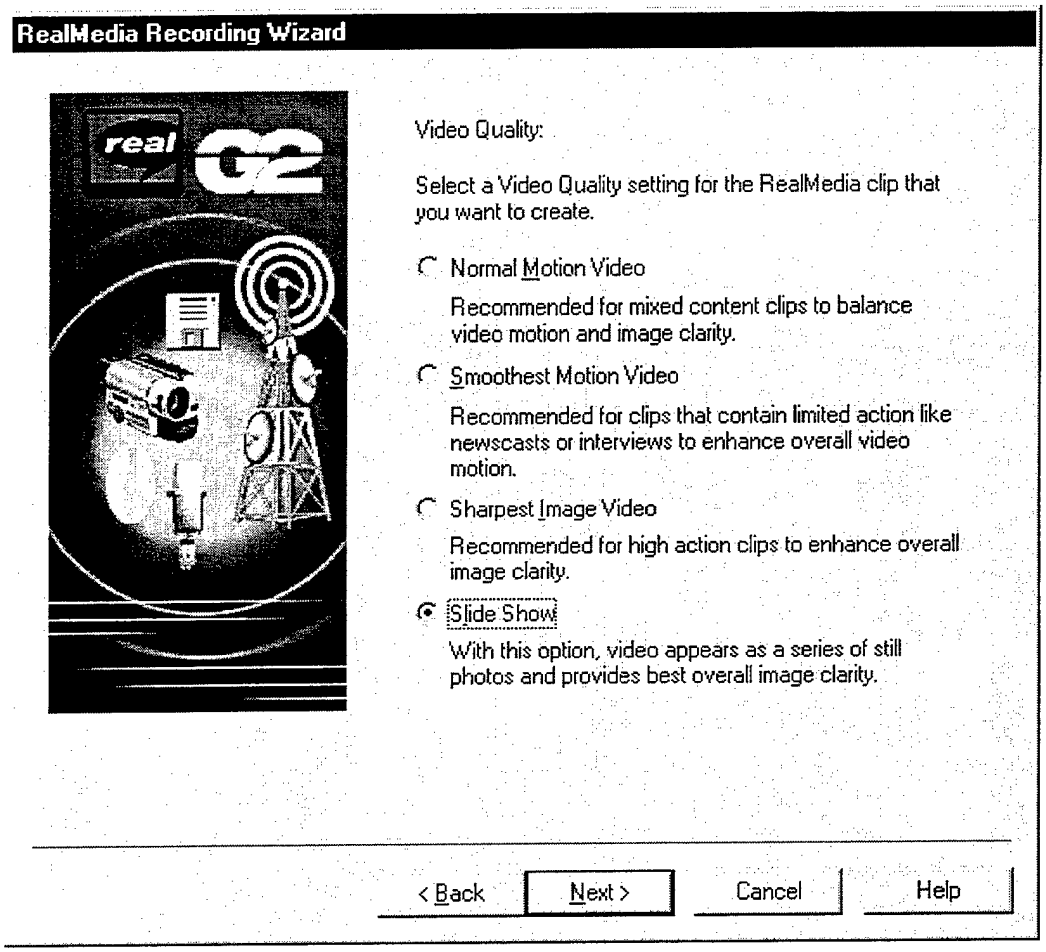

Figure 5.8: Recording Wizard – Video Quality

7. Server Settings

To complete the configuration of RealProducer G2, the media server that the live video stream will be broadcast from must be identified. This involves specifying the name and port number of the server, a filename used to reference the live video stream, and a username and password necessary to login into the server. In our implementation, the RealServer is "ratbert", the port is "4040", the filename used is "live", and the username and password are both "utips" (Figure 5.9).

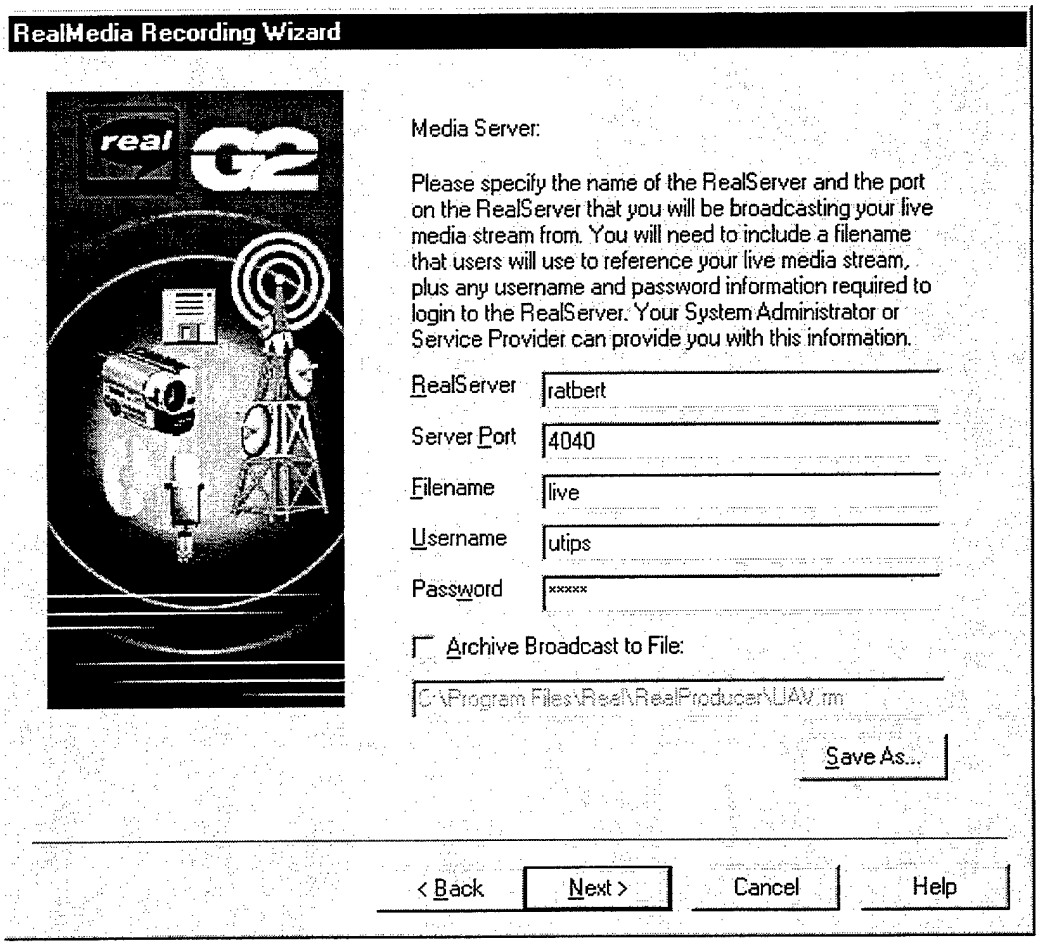

Figure 5.9: Recording Wizard – Media Server Settings

8. Configuration Complete

When configuration of RealProducer G2 is complete, the main application window (Figure 5.10) is displayed from which the user can start and stop the streaming process and make configuration changes.

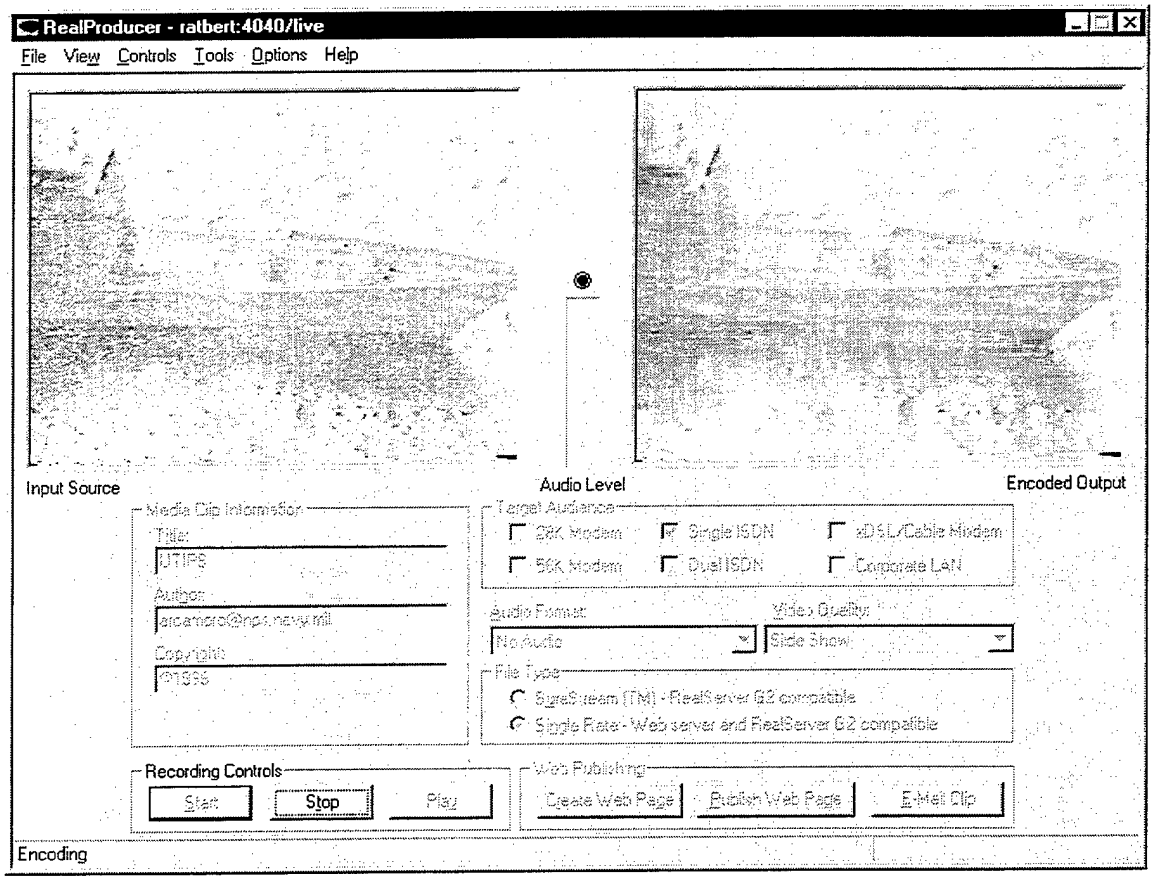

Figure 5.10: RealProducer G2 Application Window

F. STILL IMAGERY

Unlike video streaming, there is no ready to run commercial software solution for capturing still imagery. Rather, the CapturePRO ActiveX control is embedded in an application developed by the authors for this purpose. An Osprey-100 capture board provides the video source for capturing imagery and must be first be configured for operation.

1. Capture Source Settings

First, capture source settings must be initialized. This is done when the Track Injection and Image Capture System application is started. The video input, signal specification, and capture board must be selected. As shown in Figure 5.11, it is Composite 1, NTSC-M, and Board 1.

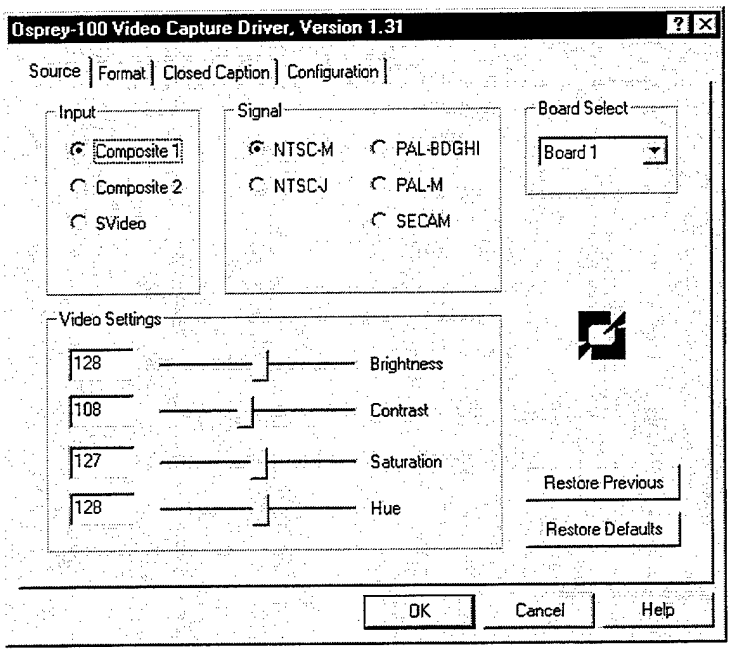

Figure 5.11: Opsrey-100 Video Capture Driver Source Settings

2. Capture Format Settings

Second, the color format and video must be set. As shown in Figure 5.12, they are RGB15 and 320x240.

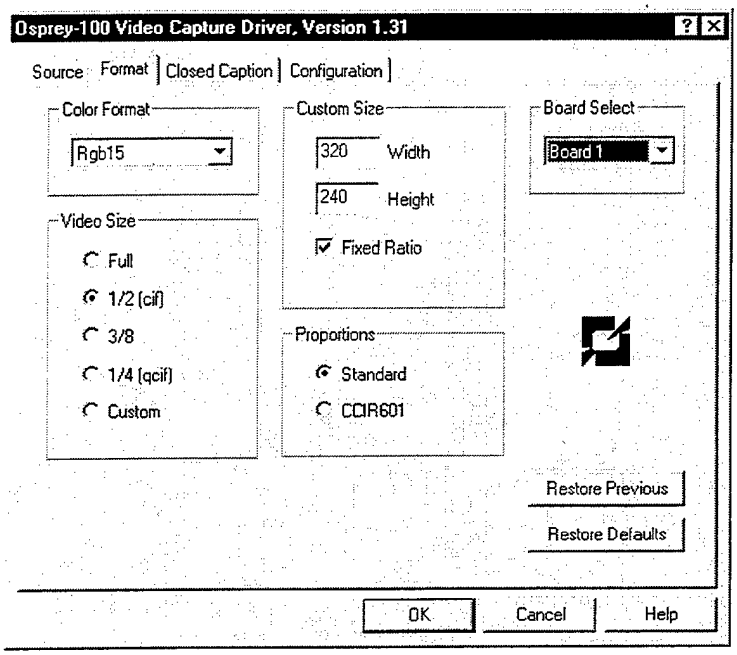

Figure 5.12: Opsrey-100 Video Capture Driver Format Settings

3. Configuration Complete

With the capture settings configured, the main application window (Figure 5.13) is displayed, and imagery is visible. To capture an image, the operator presses the "Capture Frame" button. Pressing "Adjust Video" allows the operator to adjust the video settings using the dialog shown above in Figure 5.11. Appendix A contains the code used to create the user interface.

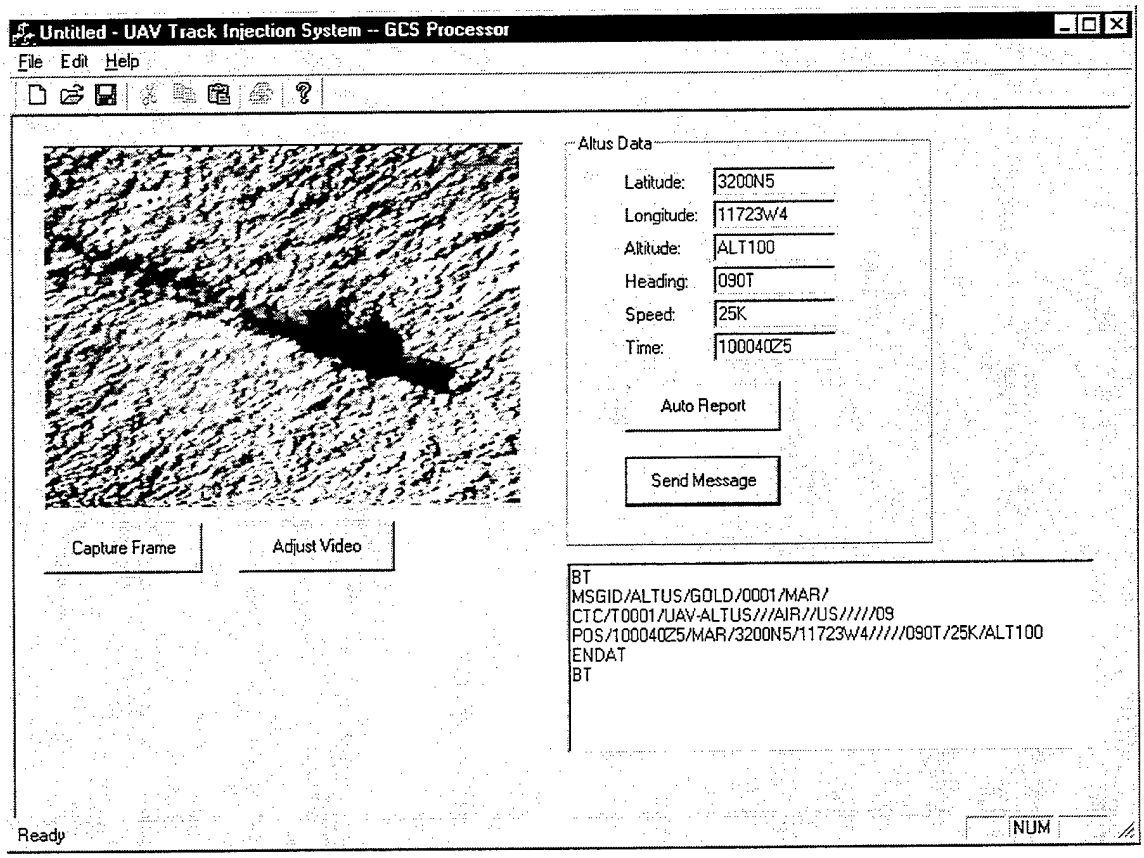

Figure 5.13: Track Injection and Image Capture Application Window

4. Capturing Still Imagery

Imagery without time and position data is not of much value to warfighters. Therefore when an image is captured, there must be a method by which this data is associated with the captured image. To accomplish this, when the operator presses the

"Capture Frame" button, the track number, the time of capture, and the position of the UAV are recorded. This data is used to generate an OTG contact report for the image and is injected into GCCS, additionally, this data is recorded in an imagery database for later retrieval using a web browser.

a. Track Number

A unique track number is assigned to the image. This track number identifies the image for correlation between GCCS and the image database.

b. Time of Capture

The date and time of capture is recorded and is used to generate a unique file name for the image and is used for the time data on the OTG message.

c. Position

As discussed in Chapter IV, position data is derived from UAV telemetry data. This serves as an approximate position of the reported image. Following capture, the next immediate telemetry packet is read and processed. This data is used as the position of the image in the report as there is currently no means of extrapolating the actual position of the image.

d. Process

The image is converted to a JPEG format image that is saved, over the network connection, on the server. The directory in which it is saved is read-only accessible to web browsers. The file name, track number, time, and position data are used to build an SQL statement that updates the imagery database.

G. DATA TRANSFER

Streaming video and still imagery are sent to the domain server using the same TCP/IP connection discussed in chapter IV. The server is then responsible for distributing imagery to requesting clients.

1. Streaming Video

RealProducer does not have the capability of serving "live" content to requesting clients therefore additional software is required. RealServer G2, also from RealNetworks and also free for evaluation purposes, provides the services necessary to serve streaming video. RealServer is installed on the domain server. The purpose of this is twofold: to minimize bandwidth demand and to maximize GCS Processor performance.

a. Bandwidth

For every client connection to RealServer, a separate – and duplicate – data stream is sent. The serial connection between the GCS Processor and the domain server is just not fast enough to support this network demand and would be quickly overwhelmed with network traffic. A network connection is required for clients. The domain server is connected to an Ethernet backbone and is the logical point from which to serve streaming video.

b. Performance

Capturing and encoding video is a CPU intensive task. Thus measures must be taken to minimize the amount of processing required by the GCS Processor. Locating RealServer on the domain server releases the GCS Processor from the burden of serving content to users.

2. Still Imagery

Still images, when captured, are sent to the domain server for storage and later retrieval using a web browser. As was the case for streaming video, images are stored on

the server to minimize bandwidth demand between the GCS Processor and the domain server resulting from web client requests.

H. DISTRIBUTION OF IMAGERY

Imagery is distributed using commercially available software. Microsoft's Internet Information Server 4.0 is used to co-ordinate the various data sources to users. These data sources are RealServer G2, which provides the streaming video, and Microsoft's SQL Server 6.5, which provides still imagery content. When viewing the UTI²PS home page (Figure 5.14), the most recently captured image is displayed as well as buttons to the "live" video stream and the imagery database.

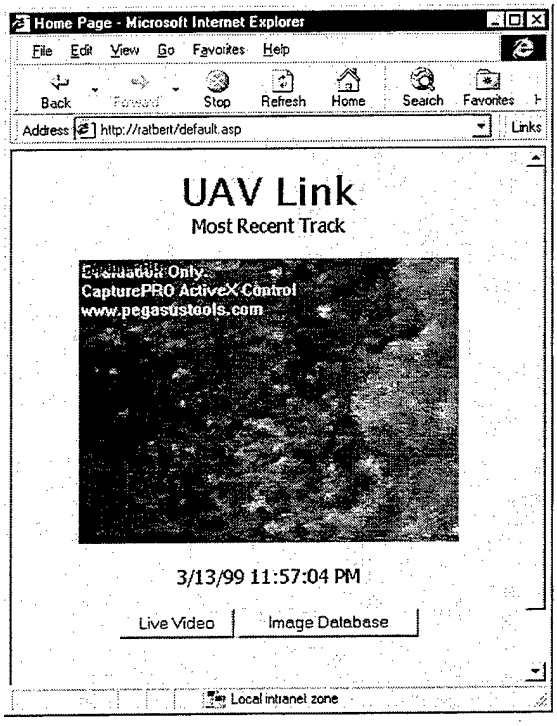

Figure 5.14: UTI²PS Home Page

1. Streaming Video

With RealServer G2 as the streaming video content provider, RealNetworks' RealPlayer G2 is required to view "live" video and must be installed on client

workstations. When a user requests the "live" feed, a new browser window is created and embedded within the window is the RealPlayer G2 ActiveX control which contains the properties required to receive the "live" video.

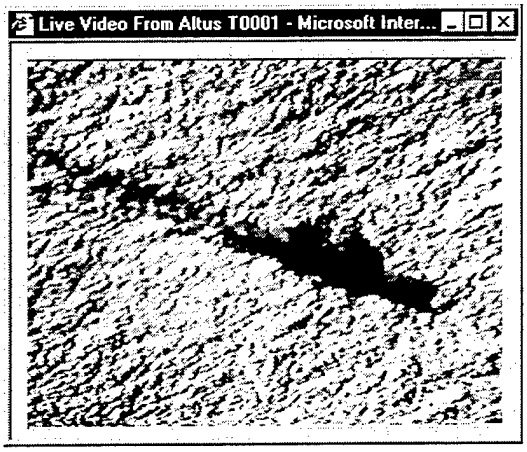

Figure 5.15: Live Video Feed from RealServer

2. Imagery Database

The imagery database is used to maintain imaging data from the UAV. It is built using Microsoft's SQL Server 6.5. The reason for using SQL Server is that it is easily scalable if network demand increases and that a 120 day evaluation version is freely available. This database contains only one table, the contents of which are: track number, image name, time of contact, latitude, longitude, altitude, and remarks. See Appendix C for the database definition.

By using IIS 4.0 and SQL Server 6.5, retrieving data from the database for presentation using web browsers is easily accomplished using Microsoft's VBScript. The Image Database web page (Figure 5.16), when retrieved, executes a script on the server that retrieves imagery data from SQL Server and is presented to the web client as a web page. Appendix D contains the HTML code for the web pages.

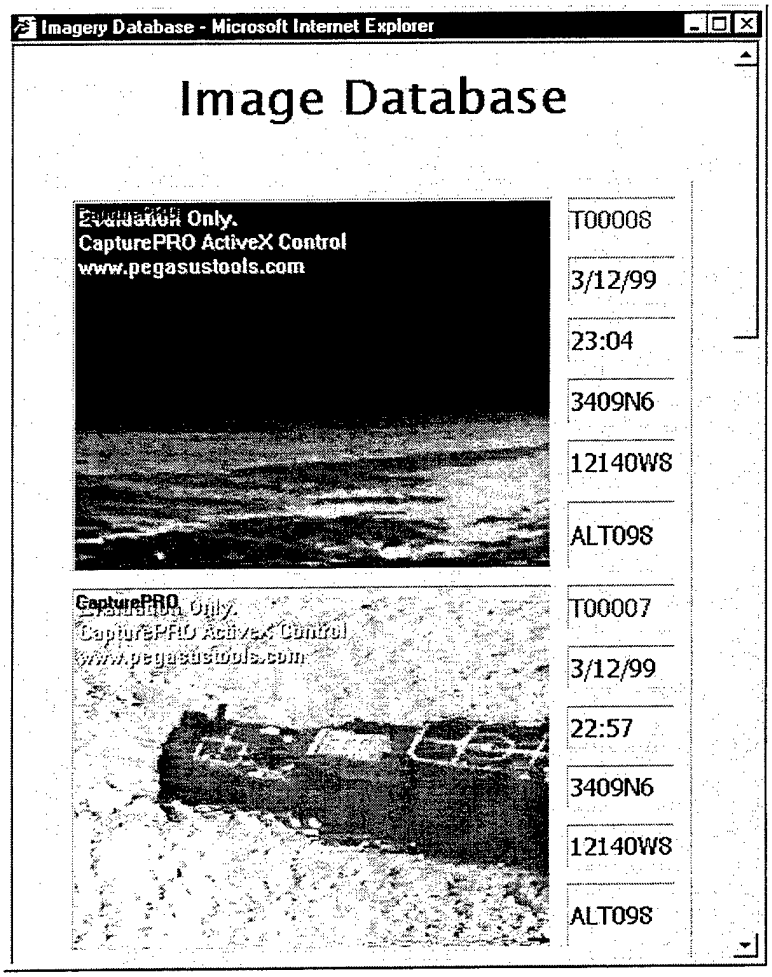

Figure 5.16: Image Database Web Page

I. SUMMARY

By using low-cost commercially available software it was demonstrated that UAV imagery can be captured and distributed to users over a TCP/IP network. Streaming video in particular is widely available on the Internet and recent advances in this field make the application in our research quite valuable. Capturing still imagery, though incorporating commercially available components, still requires a programmatic solution to provide additional data to correlate imagery with GCCS tracks.

VI. PROOF-OF-CONCEPT DEMONSTRATION

A. INTRODUCTION

This chapter describes the proof-of-concept demonstration for UTI^{2}PS in the Systems Technology Battle Lab Annex. The purpose of the demonstration is to prove the viability of distributing UAV telemetry data and imagery to the warfighter. The remainder of this chapter discusses limitations, the lab simulation, and its configuration.

B. LIMITATIONS

This section discusses three major limitations discovered during the research and implementation phases of this thesis: a disabled data source, bandwidth available from the UAV GCS and the NPS STBL, and security considerations.

1. Data Source Disabled

The first GCS software build made UAV telemetry data externally available through and RS-422 port, however, the current software does not allow telemetry data to be sent to the external GCS RS-422 serial port. The unavailability of required data from the port was unknown at the time our research began. Testing by General Atomics personnel verified that the port itself was functional, but that the port was wired incorrectly. Further testing revealed that the software module that provides telemetry data to the serial port was disabled in the current software build to provide other UAV capabilities and is expected to be corrected with the next software build.

2. Bandwidth Between CIRPAS and NPS

OTG message and imagery routing requires a TCP/IP connection. Personnel in the GCS access the NPS Campus Area Network (LAN) using POTS dial-up connections on degraded phone lines. The connection speed, which is typically less than 20,000 bps, is more than sufficient for routing OTG messages, and is an acceptable rate for the routing of an occasional still image. Streaming video, even at the lowest encoding rate

(28.8 kbps), is not possible without total degradation of the network connection. Multi-link point-to-point in which two phone lines are used in parallel was explored as a solution for increasing the available bandwidth, but with the current phone lines the maximum rate expected is no more than 40,000 bps and therefore is not a feasible solution.

As discussed in Chapter V, the optimum configuration for Real Encoder is single-channel ISDN in slide show mode. This provides updated content once per second with an acceptable resolution of 320x240 pixels. With the additional requirements for OTG messages and still images, testing reveals that 115,000 bps is the practical minimum bandwidth required. Dual-channel ISDN or 128K Frame Relay between CIRPAS and NPS is required for field testing all features.

3. Security Concerns

The ultimate goal of this research is to distribute UAV track data to GCCS users in the fleet in real time. To do this, OTG messages must be routed from the unclassified GCS processor to a secret track information database connected to the SIPRNET. Without a satisfactory security apparatus, testing with a live GCCS server is not possible.

C. LABORATORY SIMULATION

With the limitations described above, an overall "live" systems test was not possible, so elements of the system architecture were simulated as shown in Figure 6.1.

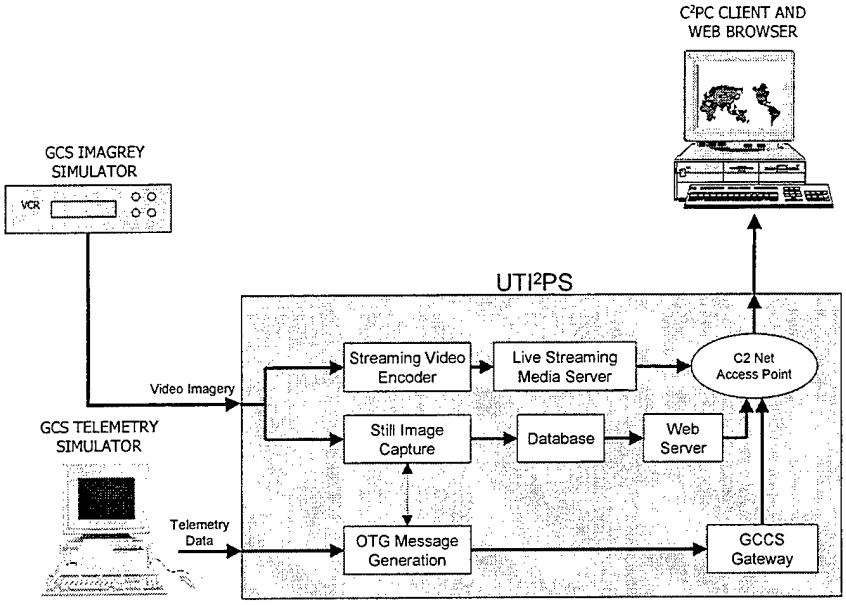

Figure 6.1: Simulated UTI²PS Architecture

1. GCS Telemetry Data

UAV telemetry data from the GCS is simulated using a telemetry data generator program (Figure 6.2) developed by the authors, which generates telemetry data packets in the same format as specified for the GCS. The user enters a starting point in latitude and longitude, and parameters for course, speed, and altitude. Actual telemetry data packets are generated at a rate of 10Hz and sent to the GCS RS-422 serial port. The purpose of the generator is to simulate sending UAV data from the GCS with which to test the OTG generation algorithm on the GCS processor. The code used to create the GCS telemetry simulator is contained in Appendix B.

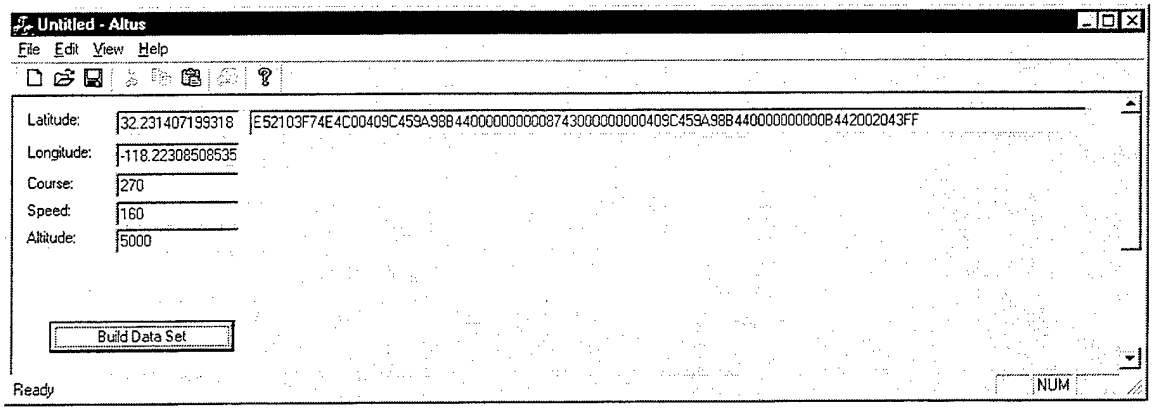

Figure 6.2: Telemetry Packet Generator

2. GCS Imagery Data

CIRPAS provided a VHS tape containing UAV imagery with which we tested still image capture and streaming video encoding using a VCR. Both the GCS and a VCR use standard NTSC analog video signals.

3. Remote Connection

Bandwidth requirements were tested using a null-modem serial cable and a remote access server. This allowed the flexibility of testing line speeds from 19,200 bps to 115,000 bps to determine the effects of available bandwidth with required bandwidth. Testing with conventional modems was impractical given the condition of the phone lines.

4. GCCS Track Information Database

Inter-National Research Institute's (INRI) C^2PC Suite for Windows NT is used as a stand-alone GCCS track information database and COP. With the C^2PC Gateway application we are able to test the processing of OTG messages generated by the GCS processor as well as display tracks on the GCCS COP.

D. CONFIGURATION

For the proof-of-concept demonstration, four Pentium II based computers were used in a networked single master domain environment. The overall actual and simulated architectures were presented earlier in Figures 3.3 and 6.1. Figure 6.3 shows how each of the links, nodes, and inter-node translations were effected and how each computer in the simulation is configured. Windows NT 4.0 was selected as the operating system because of familiarity from previous course work, programming experience, and the availability of free trial editions. Microsoft Internet Explorer 4.0 and RealNetworks Player G2 are installed on all computers. TCP/IP and NetBEUI are installed as the network transport protocols.

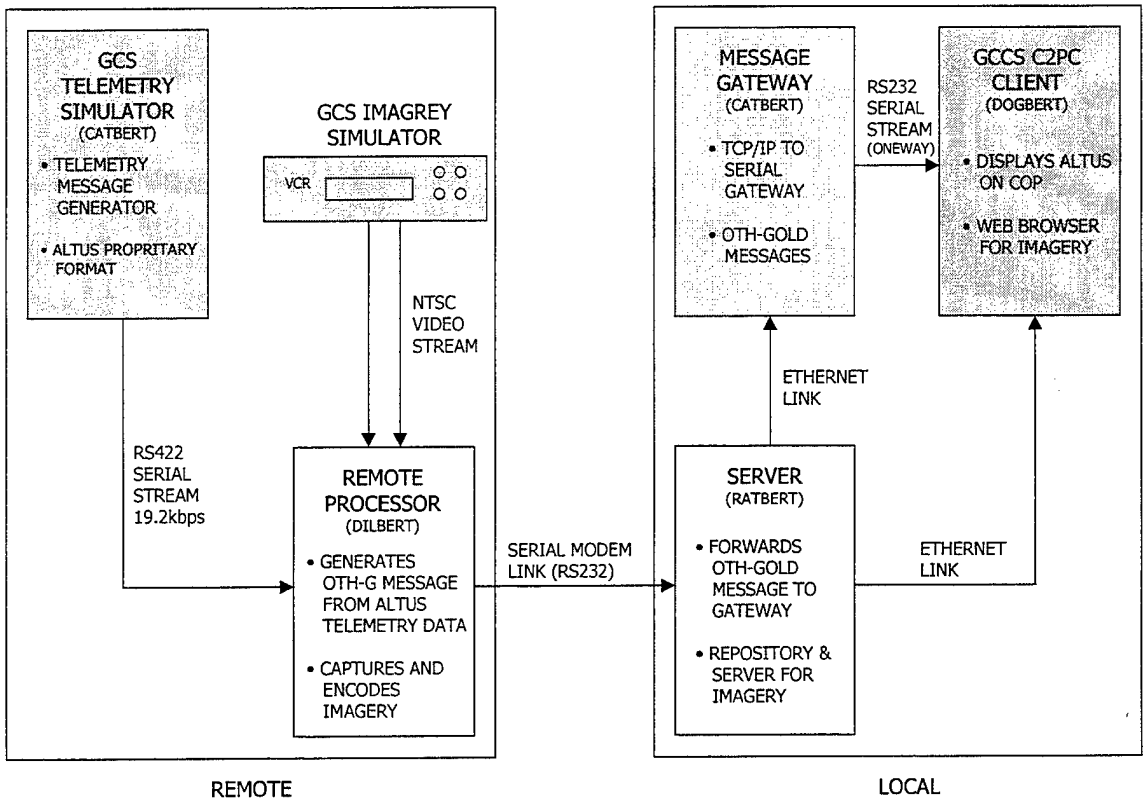

Figure 6.3: Proof-of-Concept Architecture

1. Network Server

The first computer (Ratbert, Figure 6.4) is configured as a primary domain controller for the UTIPS domain using Windows NT Server 4.0. Internet Information Server 4.0 provides WWW services with SQL Server 6.5 providing database support. RealNetworks Server G2 provides streaming media content. RAS is installed to support dial-up and direct connect serial connections between the GCS processor and the network. The network is visible to the remote connections and IP forwarding is enabled. WINS and DNS were installed to allow the use of fully qualified domain names without having to manage multiple host tables.

Ratbert

- Software
  - MS Windows NT Server 4.0
  - MS Internet Explorer 4.0
  - MS SQL Server 6.5
  - MS Internet Information Server 4.0
  - RealNetworks Real Server G2
- Hardware
  - Pentium II/400
  - 128 Meg Ram
  - 8 Gig (Mirrored) HDD
  - Fast Ethernet NIC
- Network
  - Primary Domain Controller
  - TCP/IP (192.168.1.1)
  - NetBEUI
  - Remote Access Service
  - Domain Name Service
  - Windows Internet Name Service

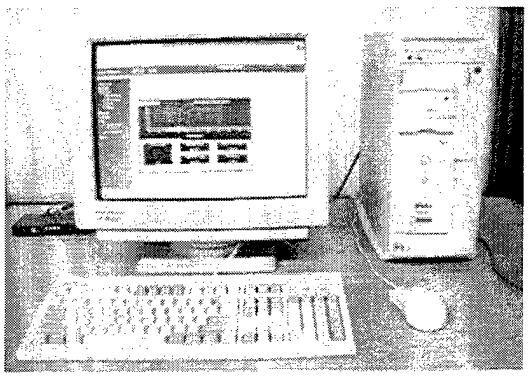

Figure 6.4: Configuration for Domain Server

2. GCS Processor

The second computer (Dilbert, Figure 6.5) is configured as the GCS processor. RealNetworks Producer G2 encodes the video signal provided by one of two video capture boards. The second video capture board is used by the Pegasus CapturePro 1.02 ActiveX control under the framework of the UAV Track Injection and Image Capture Application for still image capture. The RS-422 serial port is for receiving data from the Telemetry Data Generator. Dial-up networking is installed for connectivity to Ratbert and the UTIPS domain.

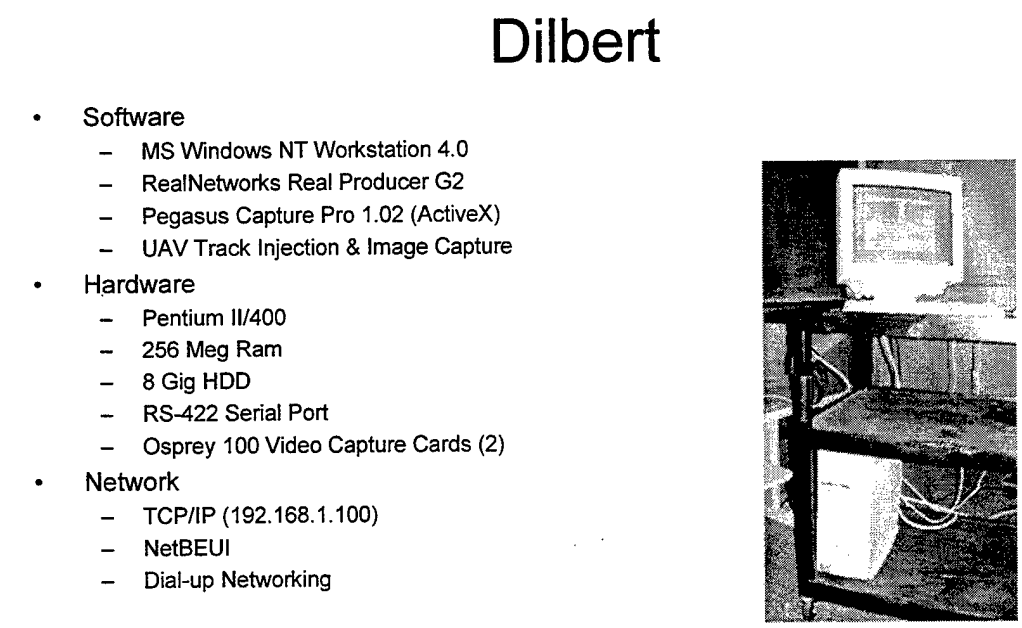

Figure 6.5: Configuration for GCS Processor

3. C²PC Gateway and Client

The third computer (Dogbert, Figure 6.6) is configured as the C²PC Gateway and client using INRI's C²PC suite. Its primary function was for testing the format of OTG messages generated by Dilbert for compatibility with GCCS. The C²PC client also provided a means to display track information.

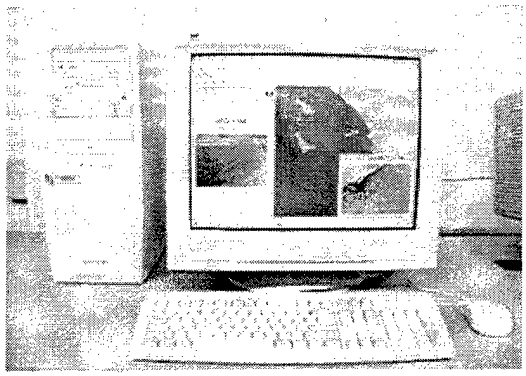

Figure 6.6: C²PC Gateway and Client

4. Telemetry Generator and GCCS Relay

The fourth computer (Catbert, Figure 6.7) was configured as the telemetry packet generator and GCCS relay. As a packet generator, telemetry data packets are sent to an RS-422 serial board. GCCS relay runs as an independent process that "listens" for OTG messages on the network, messages, when received are routed to an RS-232 serial port.

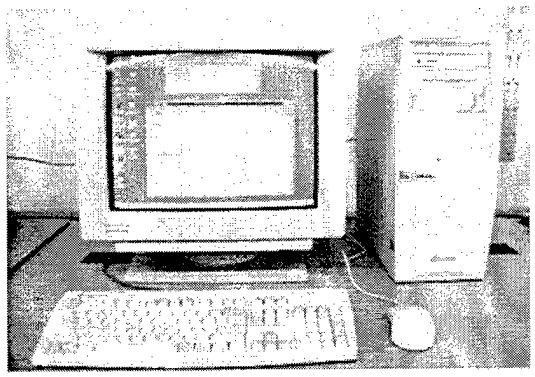

Figure 6.7: Telemetry Generator and GCCS Relay

E. RESULTS

The architecture, simulated in the lab (Figure 6.8), successfully demonstrated the viability of injecting track data and distributing imagery into a command and control network. However, the availability of bandwidth between the GCS processor and the domain server places a significant constraint on the level of service. Streaming video has the greatest demand for bandwidth and, if not managed properly, can block the transmission of OTG messages and captured still imagery.

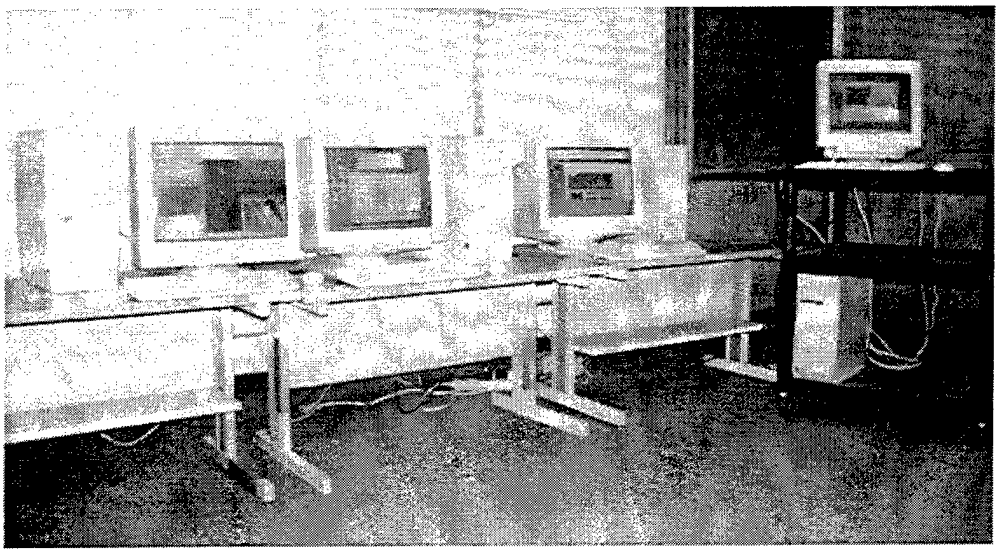

Figure 6.8: Laboratory Simulation

1. Telemetry

Altus UAV telemetry packets are properly translated to OTG contact reports and are routed correctly to the C^{2}PC Gateway and upon processing are displayed on the C^{2}PC COP. However, three issues need to be addressed.

a. OTG Message Queue

The C^{2}PC Gateway does not immediately process OTG messages upon receipt. Received messages are held in a queue and message processing does not occur

until another message is received. With the Track Injection and Image Capturing Application running in "auto" mode, OTG messages are sent every 20 seconds. This results in a minimum 20-second latency before the track is updated. This appears to be a design flaw of the C²PC Gateway itself and with frequent OTG traffic is not an issue, however with infrequent messages, there can be a substantial delay before the message is processed.

b. Time Resolution

Time, which is resolved to the minute, impacts the maximum frequency in which a track's position may be updated. C²PC will process and discard position reports with the same date-time group and track number. In the "auto" mode with OTG messages sent every 20 seconds, the first message with a new date-time group will cause the track to be updated, the following reports will be ignored. Therefore, the maximum track update frequency is one per minute.

c. Position Resolution

Latitude and longitude in the OTG message have a resolution to the nearest minute. Thus, the actual position of the Altus UAV, as well as any tracks reported through the imaging subsystem, will be rounded to the nearest minute in the OTG contact report for a position error of 0.7 nautical miles near the equator. Because UTI²PS is not designed to be a targeting system, the impact is marginal.

2. Streaming Video

Streaming video, though compressed, places the greatest demand on bandwidth. Using standard serial ports for connectivity between the GCS processor and the domain server, the maximum line speed available is 115,200 bps. With this constraint, best performance as measured in terms of frame rate and video quality was obtained with the following settings: 320x240 resolution, dual ISDN/slide show for an encoding rate of 82

kbps and a frame rate of 1.2 frames per second. Without dual channel ISDN this is an unrealistic speed.

Reducing the line speed constraint to 57,600 bps requires setting the target audience to 28.8K modem results in an encoding rate of 20 kbps and a frame rate of 0.3 frames per second. This leaves sufficient overhead to send OTG messages and still imagery. With the quality of phone lines at CIRPAS lacking, even a multi-link connection does not provide sufficient bandwidth. A bigger pipe between the GCS processor and the domain server is needed.

3. Still Imagery

Capturing still images not only provides the best resolution imagery but also injects a new track into GCCS marking an approximate location of the image. With this feature, GCCS users are alerted that the UAV has found a contact of interest. However, as there is no means of capturing the actual position of the contact, the reported position is that of the UAV at the time of capture. Without the orientation of the payload camera this results in a significant and unknown error that is wholly dependent on the altitude of the UAV.

Bandwidth for imagery is a consideration, but the Track Injection and Image Capturing Application prevents users from overloading available bandwidth by blocking capture requests while imagery is being sent.

F. SUMMARY

The UTI^{2}PS simulation successfully demonstrates the value of UAV track injection and image capturing. With off-the-shelf component it shows that it is feasible to provide UAV sensor data to shooters in the fleet at low cost.

VII. CONCLUSIONS AND FURTHER RESEARCH

A. CONCLUSIONS

While it is clear that UAVs have rapidly developed to fill critical reconnaissance needs, the CONOPS for using UAVs and underlying command and control structure has not been able to keep pace to fully utilize these new advances. This is primarily due to a lack of UAV connectivity to modern Command and Control systems. This thesis has demonstrated through a proof-of-concept laboratory implementation, how UAV telemetry information can be viably used to inject real-time tracks of a UAV's position into the Global Command and Control System's (GCCS) Common Operational Picture (COP). Methods of presenting imagery from a UAV have also been demonstrated, including both near-real-time streaming digital video, and digital still imagery from a live feed or database repository. Ways of incorporating telemetry along with digital imagery was also shown, including the injection of tracks within the GCCS COP in order to mark the location of corresponding images of interest.

The technologies developed utilize common-off-the-shelf (COTS) software and hardware, as well as open standards, to as great an extent as possible. Due to the proprietary nature of the Ground Control Station (GCS), a serial RS-422 interface was required for extraction of kinematics telemetry data. A serial RS-232 interface was required for injection of OTG messages into the GCCS C^2PC gateway. With the exception of these serial interfaces, a network-based architecture was used in order to provide high levels of availability, scalability, compatibility, and flexibility. Ideally, a network-centric system where the UAV telemetry and imagery sensors are networked directly to the Command and Control systems would be desired due to the elimination of potential bottlenecks and single points of failure. Although not currently possible, this approach should be pursued as Ground Control Stations and Command and Control systems evolve.

The CIRPAS Altus UAV was utilized as a test-bed for the development of the UTI^{2}PS system, and can subsequently be used as a research tool for the evaluation of evolving UAV Command and Control doctrine. CIRPAS also operates other aircraft, including the Pelican Optionally Piloted Vehicle (OPV) and recently acquired Predator UAV, which can also be used with the UTI^{2}PS system for further research and evaluation.

B. RECOMMENDATIONS FOR FURTHER RESEARCH

Due to limitations discovered and documented during the research and implementation of this thesis, a "live" demonstration of the UTI^{2}PS system was not possible, although the viability of the system was demonstrated through a laboratory simulation. These limitations are prime areas for further research along with other recommendations that follow.

1. Verification of GCS RS-422 Port Telemetry

As was previously discussed, in the GCS software build available at the time of this thesis, the software module that provides telemetry data to the serial RS-422 port was disabled. It is expected that this will be corrected in the next software build, which is scheduled for installation in April 1999. This should be verified for proper operation after confirmation of the re-enabling of this feature. Then a demonstration of UTI^{2}PS using telemetry and imagery data from the actual GCS can be conducted.

It was also discovered in our research that the GCS has an Ethernet network connection, the stated function of which is for the loading of new software builds. Although modifications to the GCS software would be required, investigation into the possibility of utilizing this network connection for retrieval of telemetry data warrants further research.

2. Alternative Communication Means Between the GCS and STBL

It was determined through laboratory simulation that the bandwidth between the GCS and STBL, or other GCCS / SIPRNET injection points, is a limiting factor in the

type and quality of imagery that can be presented. Thus alternative means of establishing a TCP/IP connection between the GCS and injection point should be researched. As was determined through simulation, a single modem, or even dual multi-linked modem connection would provide sufficient bandwidth for telemetry data with an occasional still digital image only, but not streaming digital imagery. To provide streaming digital imagery of moderate quality along with telemetry data for track injection, a minimum bandwidth of 115 kbps was determined to be necessary. Communication via frame relay, dual ISDN, or other means should be investigated and evaluated through proof-of-concept demonstration.

3. Connection of UTI²PS to "Live" GCCS and SIPRNET

Because of security concerns, connection of the UTI²PS system to the "live" GCCS system and SIPRNET was not conducted. In order to route OTG messages from the unclassified proof-of-concept UTI²PS system to the classified GCCS track information database requires special security concerns. A method of doing this was developed in which a one-way serial cable could be used to connect the two systems, ensuring that data could only flow one-way from the lower classified UTI²PS to the higher classified GCCS. This one-way cable mechanism was utilized in the simulated implementation, and is a viable solution. Actual implementation requires further work.

Additional research is also needed before connection to the classified SIPRNET for dissemination of imagery. A satisfactory security apparatus or firewall is necessary to ensure security concerns are satisfied before connecting the two networks of different classification.

4. Time Synchronization of Data using GPS Time

In the proof-of-concept implementation, the time annotated on all imagery and OTG messages is local system time rather than actual GPS time. GPS time is available from the kinematics telemetry data, and thus could be decoded and used for all time

stamps rather than system time. This would be necessary for accuracy in an actual implementation of UTI^{2}PS, and thus warrants further research and development.

5. Annotation / Overlay of Imagery with Telemetry Data

As was previously discussed, when a still digital image is captured, a track marking this contact of interest is injected into the GCCS COP, and recorded along with the image into a database. However, there is no means of capturing the actual position of the contact, as the reported position is that of the UAV at the time of capture. Without the orientation of the payload camera, this results in a significant and unknown error that is wholly dependent on the altitude of the UAV.

The telemetry data, which indicates the actual position of the contact of interest from the payload camera imagery, is available within the GCS. Additional research is required to determine how this data can best be extracted and incorporated within the UTI^{2}PS.

One potential solution would be to obtain the video feed by tapping off of the RGB inputs to the GCS CRTs, which contain the actual position data. A downfall of this solution is that there is also a great deal of additional information displayed on the GCS CRTs that is unnecessary for inclusion in disseminated imagery.

A new closed captioning feature is being incorporated into the GCS that will provide telemetry data for the imagery in an NTSC closed caption format. Since the OPSREY-100 video capture boards are capable of decoding closed-captioned information, this may prove to be the most viable solution.

6. Direct linking of GCCS Track Symbology to Imagery

The proof-of-concept system developed provides tracks of the UAV, as well as tracks indicating contacts of interest, on the GCCS COP. It also provides streaming digital imagery from the UAV, and still digital images of the contacts of interest, along with pertinent telemetry information and a track number that can be correlated to the track on the GCCS COP.

The development of a seamless link between the GCCS COP and imagery, so that a user can just click on a track to bring up the corresponding imagery from that location, warrants further research.

7. Upgrade of Existing UAV Architecture

The current architecture used by the UAV and GCS could be improved upon by the incorporation of modern digital imagery technology and a network centric architecture. Ideally, imagery should be captured at the UAV in digital format, such as motion JPEG, and then transmitted along with telemetry data directly to the end tactical user using a network centric architecture. This would provide the highest level of availability and thus should be pursued in further research.

LIST OF REFERENCES

1. Majewski, Lieutenant Stephen E., USN, "Naval Command and Control for Future UAVs," Master's Thesis Draft, Naval Postgraduate School (NPS), Monterey, California, March, 1999.

2. Chapman, Major William G., USAF, "Organizational Concepts for the Sensor-to-Shooter World. The Impact of Real-Time Information on Airpower Targeting," Air University Press, May 1997.

3. CIRPAS Web Page [http://web.naps.navy.mil/~cirpas/].

4. *Operational Specification for Over-The-Horizon Targeting GOLD (OS-OTG) (Rev C)*, Navy Center for Tactical Systems Interoperability, 1 August 1997.

5. Ozer, Jan, "Real Improvements," *PC Magazine*, Vol. 18, No. 3, p.54, 9 February 1999.

6. Busbee, Roger, Altus Interface Memorandum, Sandia National Laboratories, March 19,1996.

7. Ouallie, Steve, *Practical C++ Programming*, O'Reilly and Associates, September 1995.

8. *Command and Control PC User's Guide*, Inter-National Research Institute, Inc., June 1997.

9. *Microsoft Visual C++ MFC Library Reference*, Microsoft Press, 1997.

10. Kruglinsky, Shepherd, Wingo, *Programming Microsoft Visual C++ 5th Edition,* Microsoft Press, August 1998.

APPENDIX A: GCS PROCESSOR VISUAL C++ CODE

```
// stdafx.h :  include file for standard system include files,
//      or project specific include files that are used frequently, but
//      are changed infrequently
//
#if !defined(AFX_STDAFX_H__B45580FB_ADE7_11D2_9646_4A728F000000__INCLUDED_)
#define AFX_STDAFX_H__B45580FB_ADE7_11D2_9646_4A728F000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN   // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>     // MFC core and standard components
#include <afxext.h>     // MFC extensions
#include <afxole.h>     // MFC OLE classes
#include <afxodlgs.h>   // MFC OLE dialog classes
#include <afxdisp.h>    // MFC Automation classes

#ifndef _AFX_NO_DB_SUPPORT
#include <afxdb.h>                      // MFC ODBC database classes
#endif // _AFX_NO_DB_SUPPORT

#ifndef _AFX_NO_DAO_SUPPORT
#include <afxdao.h>                     // MFC DAO database classes
#endif // _AFX_NO_DAO_SUPPORT

#include <afxdtctl.h>          // MFC support for Internet
                                        // Explorer 4 Common Controls

#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>            // MFC support for Windows
                                        // Common Controls

#endif // _AFX_NO_AFXCMN_SUPPORT


#include <afxsock.h>           // MFC socket extensions


//{{AFX_INSERT_LOCATION}}

// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.


#endif // !defined(AFX_STDAFX_H__B45580FB_ADE7_11D2_9646_4A728F000000__INCLUDED_)
```

```
// stdafx.cpp : source file that includes just the standard includes

//      Thesis16Jan.pch will be the pre-compiled header
//      stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

```
// MainFrm.h : interface of the CMainFrame class

//

/////////////////////////////////////////////////////////////////////////////


#if !defined(AFX_MAINFRM_H__B45580FD_ADE7_11D2_9646_4A728F000000__INCLUDED_)
#define AFX_MAINFRM_H__B45580FD_ADE7_11D2_9646_4A728F000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnEditCommunications();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.


#endif // !defined(AFX_MAINFRM_H__B45580FD_ADE7_11D2_9646_4A728F000000__INCLUDED_)
```

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Thesis16Jan.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)
        ON_WM_CREATE()
        ON_COMMAND(ID_EDIT_COMMUNICATIONS, OnEditCommunications)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
        ID_SEPARATOR,           // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.CreateEx(this,
```

```
                                        TBSTYLE_FLAT,
                                        WS_CHILD |
                                        WS_VISIBLE |
                                        CBRS_TOP    |
                                        CBRS_GRIPPER |
                                        CBRS_TOOLTIPS |
                                        CBRS_FLYBY |
                                        CBRS_SIZE_DYNAMIC) ||
                                        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
        {

                TRACE0("Failed to create toolbar\n");

                return -1;      // fail to create

        }


        if (!m_wndStatusBar.Create(this) ||

                !m_wndStatusBar.SetIndicators(indicators,

                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }

        // TODO: Delete these three lines if you don't want the toolbar to
        //  be dockable
        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        if( !CFrameWnd::PreCreateWindow(cs) )
                return FALSE;
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG
```

```
/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers


void CMainFrame::OnEditCommunications()
{

// On the to do list!

}
```

```
// Thesis16Jan.h : main header file for the THESIS16JAN application
//

#if !defined(AFX_THESIS16JAN_H__B45580F9_ADE7_11D2_9646_4A728F000000__INCLUDED_)
#define AFX_THESIS16JAN_H__B45580F9_ADE7_11D2_9646_4A728F000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanApp:
// See Thesis16Jan.cpp for the implementation of this class
//

class CThesis16JanApp : public CWinApp
{
public:
	CThesis16JanApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CThesis16JanApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation
	//{{AFX_MSG(CThesis16JanApp)
	afx_msg void OnAppAbout();
		// NOTE - the ClassWizard will add and remove member functions
		// here.
		// DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif // !defined(AFX_THESIS16JAN_H__B45580F9_ADE7_11D2_9646_4A728F000000__INCLUDED_)
```

```
// Thesis16Jan.cpp : Defines the class behaviors for the application.

//


#include "stdafx.h"

#include "Thesis16Jan.h"

#include "MainFrm.h"
#include "Thesis16JanDoc.h"
#include "Thesis16JanView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanApp

BEGIN_MESSAGE_MAP(CThesis16JanApp, CWinApp)
        //{{AFX_MSG_MAP(CThesis16JanApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros
                // here.

                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanApp construction

CThesis16JanApp::CThesis16JanApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CThesis16JanApp object

CThesis16JanApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanApp initialization

BOOL CThesis16JanApp::InitInstance()
{
        if (!AfxSocketInit())
        {
                AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
                return FALSE;
        }

        // Initialize OLE libraries
        if (!AfxOleInit())
        {
```

```
        AfxMessageBox(IDP_OLE_INIT_FAILED);
        return FALSE;
    }

    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();    // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();     // Call this when linking to MFC
                                  // statically

#endif


    // Change the registry key under which our settings are stored.

    // TODO: You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    LoadStdProfileSettings(0);  // Load standard INI file options
                                // (including MRU)



    // Register the application's document templates.
    // Document templates serve as the connection between documents,
    // frame windows and views.


    CSingleDocTemplate* pDocTemplate;

    pDocTemplate = new CSingleDocTemplate(

        IDR_MAINFRAME,

        RUNTIME_CLASS(CThesis16JanDoc),
        RUNTIME_CLASS(CMainFrame),       // main SDI frame window
        RUNTIME_CLASS(CThesis16JanView));
    pDocTemplate->SetContainerInfo(IDR_CNTR_INPLACE);
    AddDocTemplate(pDocTemplate);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized,
    // so show and update it.

    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
```

```
	return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
		// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CThesis16JanApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanApp message handlers
```

```
// Thesis16JanView.h : interface of the CThesis16JanView class

//

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INCLUDES()

#include "capture.h"

//}}AFX_INCLUDES

#include "Structures.h"


#if !defined(AFX_THESIS16JANVIEW_H__B4558101_ADE7_11D2_9646_4A728F000000__INCLUDED_)
#define AFX_THESIS16JANVIEW_H__B4558101_ADE7_11D2_9646_4A728F000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CThesis16JanCntrItem;

class CThesis16JanView : public CFormView
{
protected: // create from serialization only
        CThesis16JanView();
        DECLARE_DYNCREATE(CThesis16JanView)

public:
        //{{AFX_DATA(CThesis16JanView)
        enum { IDD = IDD_THESIS16JAN_FORM };
        CString m_altitude;
        CString m_heading;
        CString m_latitude;
        CString m_longitude;
        CString m_speed;
        float   m_tempLat;
        float   m_tempLong;
        CString m_messageString;
        CString m_time;
        CCapture m_Capture;
        //}}AFX_DATA
        stIniTxt stIniStrings;

// Attributes
public:
        CThesis16JanDoc* GetDocument();
        // m_pSelection holds the selection to the current CThesis16JanCntrItem.
        // For many applications, such a member variable isn't adequate to
        //  represent a selection, such as a multiple selection or a selection
        //  of objects that are not CThesis16JanCntrItem objects.  This selection
        //  mechanism is provided just to help you get started.

        // TODO: replace this selection mechanism with one appropriate to your app.
        CThesis16JanCntrItem* m_pSelection;

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
```

```
        //{{AFX_VIRTUAL(CThesis16JanView)
        public:
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual BOOL DestroyWindow();
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        virtual void OnInitialUpdate(); // called first time after construct
        virtual BOOL IsSelected(const CObject* pDocItem) const;// Container support
        //}}AFX_VIRTUAL

// Implementation
public:
        CString m_month;
        int m_board;
        int m_gccs;
        int m_telemetry;
        virtual ~CThesis16JanView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CThesis16JanView)
        afx_msg void OnDestroy();
        afx_msg void OnSetFocus(CWnd* pOldWnd);
        afx_msg void OnSize(UINT nType, int cx, int cy);
        afx_msg void OnInsertObject();
        afx_msg void OnCancelEditCntr();
        afx_msg void OnStartvideo();
        afx_msg void OnGetTelemetry();
        afx_msg void OnTimer(UINT nIDEvent);
        afx_msg void OnAutoSend();
        afx_msg void OnStopSending();
        afx_msg void OnBtnCapture();
        afx_msg void OnClose();
        afx_msg void OnBtnVideo();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()

private:
        CString m_track;

        bool m_timerStatus;
        int m_nTimer;
        int m_nCount;
        int messageCounter;

        enum { nMaxCount = 10000 };

        void CThesis16JanView::ResizeWindow();

        CString CThesis16JanView::GetSerial();
        CString CThesis16JanView::GetSerial(CTime);
        bool CThesis16JanView::SetSerial(CString, CString);

};
#ifndef _DEBUG  // debug version in Thesis16JanView.cpp
inline CThesis16JanDoc* CThesis16JanView::GetDocument()
   { return (CThesis16JanDoc*)m_pDocument; }
#endif
```

```
/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_THESIS16JANVIEW_H__B4558101_ADE7_11D2_9646_4A728F000000__INCLUDED_)
```

```
// Thesis16JanView.cpp : implementation of the CThesis16JanView class
//

#include "stdafx.h"
#include "Thesis16Jan.h"
#include "Thesis16JanDoc.h"
#include "CntrItem.h"
#include "Thesis16JanView.h"
#include "DialogSettings.h"
#include "DlgHostname.h"
#include "CMsgGold.h"
#include "CTelemetry.h"
#include "Float.h"
#include "Structures.h"
#include <math.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CMsgGold cmsgGold;
CTelemetry cTelemetry;
StrCSTm SCSTdata;
int iTrack;

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanView

IMPLEMENT_DYNCREATE(CThesis16JanView, CFormView)

BEGIN_MESSAGE_MAP(CThesis16JanView, CFormView)
        //{{AFX_MSG_MAP(CThesis16JanView)
        ON_WM_DESTROY()
        ON_WM_SETFOCUS()
        ON_WM_SIZE()
        ON_COMMAND(ID_OLE_INSERT_NEW, OnInsertObject)
        ON_COMMAND(ID_CANCEL_EDIT_CNTR, OnCancelEditCntr)
        ON_BN_CLICKED(IDC_STARTVIDEO, OnStartvideo)
        ON_BN_CLICKED(IDC_GET_TELEMETRY, OnGetTelemetry)
        ON_WM_TIMER()
        ON_BN_CLICKED(IDC_AUTO_SEND, OnAutoSend)
        ON_BN_CLICKED(IDC_STOP_SENDING, OnStopSending)
        ON_BN_CLICKED(IDC_BTN_CAPTURE, OnBtnCapture)
        ON_WM_CLOSE()
        ON_BN_CLICKED(IDC_BTN_VIDEO, OnBtnVideo)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanView construction/destruction

CThesis16JanView::CThesis16JanView()
        : CFormView(CThesis16JanView::IDD)
{
        //{{AFX_DATA_INIT(CThesis16JanView)
        m_altitude = _T("");
        m_heading = _T("");
        m_latitude = _T("");
        m_longitude = _T("");
        m_speed = _T("");
        m_tempLat = 0.0f;
        m_tempLong = 0.0f;
```

```
        m_messageString = _T("");
        m_time = _T("");
        //}}AFX_DATA_INIT
        m_pSelection = NULL;
        iTrack = 5;
}

CThesis16JanView::~CThesis16JanView()
{
}

void CThesis16JanView::DoDataExchange(CDataExchange* pDX)
{
        CFormView::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CThesis16JanView)
        DDX_Text(pDX, IDC_ALTITUDE, m_altitude);
        DDX_Text(pDX, IDC_HEADING, m_heading);
        DDX_Text(pDX, IDC_LATITUDE, m_latitude);
        DDX_Text(pDX, IDC_LONGITUDE, m_longitude);
        DDX_Text(pDX, IDC_SPEED, m_speed);
        DDX_Text(pDX, IDC_MESSAGE, m_messageString);
        DDX_Text(pDX, IDC_TIME, m_time);
        DDX_Control(pDX, IDC_CAPTUREPRO, m_Capture);
        //}}AFX_DATA_MAP
}

BOOL CThesis16JanView::PreCreateWindow(CREATESTRUCT& cs)
{
        return CFormView::PreCreateWindow(cs);
}

void CThesis16JanView::OnInitialUpdate()
{

// Initialize the form
        CFormView::OnInitialUpdate();
        GetParentFrame()->RecalcLayout();
        ResizeParentToFit();

        CDialogSettings dlgSettings;
        int testValue = dlgSettings.DoModal(&stIniStrings);

// Set comm routes
        m_gccs = (testValue & 0xF);
        m_telemetry = (testValue & 0xF0)/16;
        m_board = (testValue & 0xF00)/256;

        cmsgGold.CSRelay = stIniStrings.csFQDN;
        cmsgGold.SetCommPort(m_gccs);
        cTelemetry.SetCommPort(m_telemetry);

// Initialize Capture Board
        m_Capture.Connect(m_board);
        m_Capture.ShowVideoFormatDlg();
        cmsgGold.LoadString(1);
        m_timerStatus = false;
        ResizeWindow();

}

void CThesis16JanView::OnDestroy()
{
        // Deactivate the item on destruction; this is important
        // when a splitter view is being used.
```

```
    m_Capture.Disconnect();
    CFormView::OnDestroy();
    COleClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
    if (pActiveItem != NULL && pActiveItem->GetActiveView() == this)
    {
        pActiveItem->Deactivate();
        ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);
    }
}

/////////////////////////////////////////////////////////////////////////////
// OLE Client support and commands

BOOL CThesis16JanView::IsSelected(const CObject* pDocItem) const
{
        // The implementation below is adequate if your selection consists of
        //  only CThesis16JanCntrItem objects.  To handle different selection
        //  mechanisms, the implementation here should be replaced.

        // TODO: implement this function that tests for a selected OLE client item

        return pDocItem == m_pSelection;
}

void CThesis16JanView::OnInsertObject()
{
        // Invoke the standard Insert Object dialog box to obtain information
        //  for new CThesis16JanCntrItem object.
        COleInsertDialog dlg;
        if (dlg.DoModal() != IDOK)
                return;

        BeginWaitCursor();

        CThesis16JanCntrItem* pItem = NULL;
        TRY
        {
                // Create new item connected to this document.
                CThesis16JanDoc* pDoc = GetDocument();
                ASSERT_VALID(pDoc);
                pItem = new CThesis16JanCntrItem(pDoc);
                ASSERT_VALID(pItem);

                // Initialize the item from the dialog data.
                if (!dlg.CreateItem(pItem))
                        AfxThrowMemoryException();  // any exception will do
                ASSERT_VALID(pItem);

        if (dlg.GetSelectionType() == COleInsertDialog::createNewItem)
                        pItem->DoVerb(OLEIVERB_SHOW, this);

                ASSERT_VALID(pItem);

                m_pSelection = pItem;    // set selection to last inserted item
                pDoc->UpdateAllViews(NULL);
        }
        CATCH(CException, e)
        {
                if (pItem != NULL)
                {
                        ASSERT_VALID(pItem);
                        pItem->Delete();
```

```
            }
            AfxMessageBox(IDP_FAILED_TO_CREATE);
        }
        END_CATCH

        EndWaitCursor();
}

// The following command handler provides the standard keyboard
//  user interface to cancel an in-place editing session.  Here,
//  the container (not the server) causes the deactivation.
void CThesis16JanView::OnCancelEditCntr()
{
        // Close any in-place active item on this view.
        COleClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
        if (pActiveItem != NULL)
        {
                pActiveItem->Close();
        }
        ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);
}

// Special handling of OnSetFocus and OnSize are required for a container
//  when an object is being edited in-place.
void CThesis16JanView::OnSetFocus(CWnd* pOldWnd)
{
        COleClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
        if (pActiveItem != NULL &&
                pActiveItem->GetItemState() == COleClientItem::activeUIState)
        {
                // need to set focus to this item if it is in the same view
                CWnd* pWnd = pActiveItem->GetInPlaceWindow();
                if (pWnd != NULL)
                {
                        pWnd->SetFocus();   // don't call the base class
                        return;
                }
        }

        CFormView::OnSetFocus(pOldWnd);
}

void CThesis16JanView::OnSize(UINT nType, int cx, int cy)
{
        CFormView::OnSize(nType, cx, cy);
        COleClientItem* pActiveItem = GetDocument()->GetInPlaceActiveItem(this);
        if (pActiveItem != NULL)
                pActiveItem->SetItemRects();
}

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanView diagnostics

#ifdef _DEBUG
void CThesis16JanView::AssertValid() const
```

```
{
	CFormView::AssertValid();
}

void CThesis16JanView::Dump(CDumpContext& dc) const
{
	CFormView::Dump(dc);
}

CThesis16JanDoc* CThesis16JanView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CThesis16JanDoc)));
	return (CThesis16JanDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanView message handlers

void CThesis16JanView::OnStartvideo()
{
	//	Show video source dialog
	m_Capture.ShowVideoSourceDlg();

}

void CThesis16JanView::OnGetTelemetry()

{
	int iAttempts = 0;
	if (!m_timerStatus)
	{
		GetDlgItem(IDC_GET_TELEMETRY)->EnableWindow(false);
	}
	cTelemetry.OpenCommPort();
	while (!cTelemetry.ReadTelemetry())
	{
		if (iAttempts++ > 10)
		{
			AfxMessageBox ("Communications Timed Out...check data connection");
			cTelemetry.CloseCommPort();
			if (m_timerStatus)
			{
				KillTimer(1);
				m_timerStatus = false;
				SetDlgItemText(IDC_AUTO_SEND, "Auto Send");
			}
			GetDlgItem(IDC_GET_TELEMETRY)->EnableWindow(true);
			return;
		}
	}
	cTelemetry.CloseCommPort();
	if (cTelemetry.ProcessTelemetry(&SCSTdata))
	{
		m_latitude = SCSTdata.csLatitude;
		m_longitude = SCSTdata.csLongitude;
		m_altitude = SCSTdata.csAltitude;
		m_heading = SCSTdata.csHeading;
		m_speed = SCSTdata.csSpeed;
		m_time = SCSTdata.csTime;
		m_month = SCSTdata.csMonth;
	}
	else
```

```
        {
                OnGetTelemetry();
                return;
        }

        cmsgGold.LoadData(&SCSTdata);
        cmsgGold.LoadString(1);
        m_messageString = cmsgGold.MakeMsgGold();
        UpdateData(false);
        if (!m_timerStatus)
        {
                GetDlgItem(IDC_GET_TELEMETRY)->EnableWindow(true);
        }
}

void CThesis16JanView::OnTimer(UINT nIDEvent)
{
        CThesis16JanView::OnGetTelemetry();
}

void CThesis16JanView::OnAutoSend()
{

        if (!m_timerStatus)
        {
                GetDlgItem(IDC_GET_TELEMETRY)->EnableWindow(false);
                m_nTimer = SetTimer(1,20000, NULL);
                ASSERT(m_nTimer != 0);
                m_timerStatus = true;
                SetDlgItemText(IDC_AUTO_SEND, "Stop Auto");
        }
        else
        {
                GetDlgItem(IDC_GET_TELEMETRY)->EnableWindow(true);
                KillTimer(1);
                m_timerStatus = false;
                SetDlgItemText(IDC_AUTO_SEND, "Auto Report");
        }
}


void CThesis16JanView::OnStopSending()
{
        MessageBox ("Not implemented");
}

void CThesis16JanView::OnBtnCapture()
{

        GetDlgItem(IDC_BTN_CAPTURE)->EnableWindow(false);

        CDatabase CThesisData;
        CString CSQLStmt;
        CString imageName;
        CString imageTime;
        CString altusName = "\\\\ratbert\\data\\inetpub\\wwwroot\\images\\";
        CTime cTime;

        UpdateData(false);

        cTime = CTime::GetCurrentTime();
```

```
        imageName.Format("%d", cTime);
        imageName += ".jpg";
        altusName += imageName;
        imageTime = cTime.FormatGmt("%H:%M:%S %B %d %Y");
        m_Capture.SetFrameFile(altusName);
        m_Capture.CaptureFrame();

        OnGetTelemetry();

        cmsgGold.LoadString(2);
        CString l_track = cmsgGold.LoadData(m_latitude, m_longitude, m_time, m_month);
        cmsgGold.MakeMsgGold();

        CSQLStmt = "INSERT INTO tblImages ";
        CSQLStmt += "(fldImageName, fldDateTime, fldLatitude,
                    fldLongitude, fldAltitude, fldTrack) ";
        CSQLStmt += "VALUES ('";
        CSQLStmt += imageName + "', '";
        CSQLStmt += imageTime + "', '";
        CSQLStmt += m_latitude + "', '";
        CSQLStmt += m_longitude + "', '";
        CSQLStmt += m_altitude + "', '";
        CSQLStmt += l_track + "')";

        CThesisData.Open("altus", FALSE, FALSE, "ODBC;UID=altusadmin;PWD=altus");
        CThesisData.ExecuteSQL(CSQLStmt);
        CThesisData.Close();

        GetDlgItem(IDC_BTN_CAPTURE)->EnableWindow(true);

}

void CThesis16JanView::ResizeWindow()

{

        CRect lpRect;
        int iYAxis;
        int iRSide;

        GetDlgItem(IDC_CAPTUREPRO)->SetWindowPos(NULL, 20, 20, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_CAPTUREPRO)->GetClientRect(lpRect);
        iYAxis = lpRect.bottom + 30;
        iRSide = lpRect.right + 50;

        GetDlgItem(IDC_BTN_CAPTURE)->SetWindowPos(NULL, 20, iYAxis, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_BTN_VIDEO)->SetWindowPos(NULL, 150, iYAxis, 0, 0, SWP_NOSIZE);

        GetDlgItem(IDC_GROUP)->SetWindowPos(NULL, iRSide, 14, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_LABEL1)->SetWindowPos(NULL, iRSide + 40, 40, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_LABEL2)->SetWindowPos(NULL, iRSide + 40, 62, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_LABEL3)->SetWindowPos(NULL, iRSide + 40, 84, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_LABEL4)->SetWindowPos(NULL, iRSide + 40, 106, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_LABEL5)->SetWindowPos(NULL, iRSide + 40, 128, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_LABEL6)->SetWindowPos(NULL, iRSide + 40, 150, 0, 0, SWP_NOSIZE);

        GetDlgItem(IDC_LATITUDE)->SetWindowPos(NULL, iRSide + 100, 37, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_LONGITUDE)->SetWindowPos(NULL, iRSide + 100, 59, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_ALTITUDE)->SetWindowPos(NULL, iRSide + 100, 81, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_HEADING)->SetWindowPos(NULL, iRSide + 100, 103, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_SPEED)->SetWindowPos(NULL, iRSide + 100, 125, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_TIME)->SetWindowPos(NULL, iRSide + 100, 147, 0, 0, SWP_NOSIZE);

        GetDlgItem(IDC_AUTO_SEND)->SetWindowPos(NULL, iRSide + 40, 180, 0, 0, SWP_NOSIZE);
```

```
        GetDlgItem(IDC_GET_TELEMETRY)->SetWindowPos(NULL, iRSide + 40, 230, 0, 0,
SWP_NOSIZE);
        GetDlgItem(IDC_MESSAGE)->SetWindowPos(NULL, iRSide, 300, 0, 0, SWP_NOSIZE);
        GetDlgItem(IDC_BTN_CAPTURE)->ShowWindow(true);
        GetDlgItem(IDC_GET_TELEMETRY)->SetFocus();
        ResizeParentToFit();
}

CString CThesis16JanView::GetSerial(CTime MessageTime)
{
        COleVariant coleMessageID;
        CString csMessageID;
        CString csMessageTime;
        csMessageTime = MessageTime.FormatGmt("%H:%M:%S %B %d %Y");
        AfxMessageBox(csMessageTime);
        int iMessageID;

        CString sqlStmt;
        sqlStmt = "SELECT * FROM tblAltusMessages ";
        sqlStmt += "WHERE fldTime IN ";
        sqlStmt += "(SELECT MAX(fldTime) FROM tblAltusMessages)";

        CDaoDatabase altus;
        CDaoRecordset altusRecord;

        altus.Open("e:\\thesisproject\\thesis16jan\\debug\\altus.mdb",FALSE,FALSE);
        altusRecord.m_pDatabase = &altus;

        altusRecord.Open(dbOpenDynaset, sqlStmt);

        coleMessageID = altusRecord.GetFieldValue(0);
        csMessageID = (LPCSTR)(coleMessageID.bstrVal);
        iMessageID = atoi(csMessageID) + 1;
        csMessageID.Format("%04d", iMessageID);

        altusRecord.Close();

        sqlStmt = "INSERT INTO tblAltusMessages VALUES ('";
        sqlStmt += csMessageID;
        sqlStmt += "', #";
        sqlStmt += csMessageTime;
        sqlStmt += "#)";

        AfxMessageBox(sqlStmt);

        altus.Execute(sqlStmt);

        altus.Close();
        return csMessageID;
}

CString CThesis16JanView::GetSerial()
{
        COleVariant coleMessageID;
        CString csMessageID;
        CString sqlStmt;

        sqlStmt = "SELECT * FROM tblAltusMessages ";
        sqlStmt += "WHERE fldTime IN ";
        sqlStmt += "(SELECT MAX(fldTime) FROM tblAltusMessages)";

        CDaoDatabase altus;
        CDaoRecordset altusRecord;
```

```
        altus.Open(".\\altus.mdb",FALSE,FALSE);
        altusRecord.m_pDatabase = &altus;

        altusRecord.Open(dbOpenDynaset, sqlStmt);
        altusRecord.MoveFirst();

        coleMessageID = altusRecord.GetFieldValue(0);
        csMessageID = (LPCSTR)(coleMessageID.bstrVal);

        altusRecord.Close();
        altus.Close();
        return csMessageID;
}

bool CThesis16JanView::SetSerial(CString msgSerial, CString msgTime)
{

        CDaoDatabase altus;
        CString sqlStmt;

        sqlStmt = "INSERT INTO tblAltusMessages VALUES '";
        sqlStmt += msgSerial;
        sqlStmt += "', #";
        sqlStmt += msgTime;
        sqlStmt += "#)";

        altus.Open("e:\\thesisproject\\thesis16jan\\debug\\altus.mdb",FALSE,FALSE);
        altus.Execute(sqlStmt);

        altus.Close();

        return true;

}

BOOL CThesis16JanView::DestroyWindow()
{

        return CFormView::DestroyWindow();
}

CString ComputeChecksum (CString cString)
{
        CString csReturn;
        int iPtr = 0;
        int iCheckSum = 0;

        for (iPtr = 0; iPtr < 6; iPtr++)
        {
                iCheckSum += atoi(cString.Mid(iPtr, 1));
        }
        csReturn.Format("%d", iCheckSum);
        csReturn = csReturn.Right(1);
        return csReturn;
}


void CThesis16JanView::OnClose()
{
        CFormView::OnClose();
}

void CThesis16JanView::OnBtnVideo()
{
```

```
        m_Capture.ShowVideoSourceDlg();
}
```

```
// Thesis16JanDoc.h : interface of the CThesis16JanDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_THESIS16JANDOC_H__B45580FF_ADE7_11D2_9646_4A728F000000__INCLUDED_)
#define AFX_THESIS16JANDOC_H__B45580FF_ADE7_11D2_9646_4A728F000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


class CThesis16JanDoc : public COleDocument
{
protected: // create from serialization only
        CThesis16JanDoc();
        DECLARE_DYNCREATE(CThesis16JanDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CThesis16JanDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CThesis16JanDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CThesis16JanDoc)
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_THESIS16JANDOC_H__B45580FF_ADE7_11D2_9646_4A728F000000__INCLUDED_)
```

```
// Thesis16JanDoc.cpp : implementation of the CThesis16JanDoc class
//

#include "stdafx.h"
#include "Thesis16Jan.h"
#include "Thesis16JanDoc.h"
#include "CntrItem.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanDoc

IMPLEMENT_DYNCREATE(CThesis16JanDoc, COleDocument)

BEGIN_MESSAGE_MAP(CThesis16JanDoc, COleDocument)
        //{{AFX_MSG_MAP(CThesis16JanDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
        // Enable default OLE container implementation
        ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, COleDocument::OnUpdatePasteMenu)
        ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE_LINK, COleDocument::OnUpdatePasteLinkMenu)
        ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_CONVERT, COleDocument::OnUpdateObjectVerbMenu)
        ON_COMMAND(ID_OLE_EDIT_CONVERT, COleDocument::OnEditConvert)
        ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_LINKS, COleDocument::OnUpdateEditLinksMenu)
        ON_COMMAND(ID_OLE_EDIT_LINKS, COleDocument::OnEditLinks)
        ON_UPDATE_COMMAND_UI_RANGE(ID_OLE_VERB_FIRST, ID_OLE_VERB_LAST,
                                COleDocument::OnUpdateObjectVerbMenu)

        ON_COMMAND(ID_FILE_SEND_MAIL, OnFileSendMail)
        ON_UPDATE_COMMAND_UI(ID_FILE_SEND_MAIL, OnUpdateFileSendMail)

END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// CThesis16JanDoc construction/destruction

CThesis16JanDoc::CThesis16JanDoc()
{
        // Use OLE compound files
        EnableCompoundFile();

        // TODO: add one-time construction code here

}

CThesis16JanDoc::~CThesis16JanDoc()
{
}

BOOL CThesis16JanDoc::OnNewDocument()
{
        if (!COleDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)
```

```
	return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CThesis16JanDoc serialization

void CThesis16JanDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
	else
	{
		// TODO: add loading code here
	}

	// Calling the base class COleDocument enables serialization
	//  of the container document's COleClientItem objects.
	COleDocument::Serialize(ar);
}

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanDoc diagnostics

#ifdef _DEBUG
void CThesis16JanDoc::AssertValid() const
{
	COleDocument::AssertValid();
}

void CThesis16JanDoc::Dump(CDumpContext& dc) const
{
	COleDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanDoc commands
```

```
// Data structure definitions
// Structures.h


#if !defined STRUCTURES

#define STRUCTURES



struct stIniTxt

{
        CString csFQDN;
        CString csImagePath;
};

struct StrRawTm
{
        float fltLatitude;
        float fltLongitude;
        float fltAltitude;
        float fltTimeOfFix;
        float fltPitch;
        float fltRoll;
        float fltHeading;
        float fltGLatitude;
        float fltGLongitude;
        float fltGAltitude;
        float fltGTimeOfFix;
        float fltVGPitch;
        float fltVGRoll;
        float fltMHeading;
        float fltSpeed;
};

struct StrCSTm
{
        CString csLatitude;
        CString csLongitude;
        CString csAltitude;
        CString csHeading;
        CString csSpeed;
        CString csTime;
        CString csMonth;
};

struct StrFltTm
{
        float fltLatitude;
        float fltLongitude;
        float fltAltitude;
        float fltHeading;
        float fltSpeed;
        float fltTime;
};

#endif
```

```
// Font.h
//
#if !defined(AFX_FONT_H__B455810C_ADE7_11D2_9646_4A728F000000__INCLUDED_)
#define AFX_FONT_H__B455810C_ADE7_11D2_9646_4A728F000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Machine generated IDispatch wrapper class(es) created by Microsoft Visual C++

// NOTE: Do not modify the contents of this file.  If this class is regenerated by
//  Microsoft Visual C++, your modifications will be overwritten.

/////////////////////////////////////////////////////////////////////////////
// COleFont wrapper class

class COleFont : public COleDispatchDriver
{
public:
        COleFont() {}          // Calls COleDispatchDriver default constructor
        COleFont(LPDISPATCH pDispatch) : COleDispatchDriver(pDispatch) {}
        COleFont(const COleFont& dispatchSrc) : COleDispatchDriver(dispatchSrc) {}

// Attributes
public:
        CString GetName();
        void SetName(LPCTSTR);
        CY GetSize();
        void SetSize(const CY&);
        BOOL GetBold();
        void SetBold(BOOL);
        BOOL GetItalic();
        void SetItalic(BOOL);
        BOOL GetUnderline();
        void SetUnderline(BOOL);
        BOOL GetStrikethrough();
        void SetStrikethrough(BOOL);
        short GetWeight();
        void SetWeight(short);
        short GetCharset();
        void SetCharset(short);

// Operations
public:
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_FONT_H__B455810C_ADE7_11D2_9646_4A728F000000__INCLUDED_)
```

```
// font.cpp
// Machine generated IDispatch wrapper class(es) created by Microsoft Visual C++
// NOTE: Do not modify the contents of this file.  If this class is regenerated by
//  Microsoft Visual C++, your modifications will be overwritten.


#include "stdafx.h"
#include "font.h"


/////////////////////////////////////////////////////////////////////////////
// COleFont properties

CString COleFont::GetName()
{
        CString result;
        GetProperty(0x0, VT_BSTR, (void*)&result);
        return result;
}

void COleFont::SetName(LPCTSTR propVal)
{
        SetProperty(0x0, VT_BSTR, propVal);
}

CY COleFont::GetSize()
{
        CY result;
        GetProperty(0x2, VT_CY, (void*)&result);
        return result;
}

void COleFont::SetSize(const CY& propVal)
{
        SetProperty(0x2, VT_CY, &propVal);
}

BOOL COleFont::GetBold()
{
        BOOL result;
        GetProperty(0x3, VT_BOOL, (void*)&result);
        return result;
}

void COleFont::SetBold(BOOL propVal)
{
        SetProperty(0x3, VT_BOOL, propVal);
}

BOOL COleFont::GetItalic()
{
        BOOL result;
        GetProperty(0x4, VT_BOOL, (void*)&result);
        return result;
}

void COleFont::SetItalic(BOOL propVal)
{
        SetProperty(0x4, VT_BOOL, propVal);
}

BOOL COleFont::GetUnderline()
{
        BOOL result;
```

```
        GetProperty(0x5, VT_BOOL, (void*)&result);
        return result;
}

void COleFont::SetUnderline(BOOL propVal)
{
        SetProperty(0x5, VT_BOOL, propVal);
}

BOOL COleFont::GetStrikethrough()
{
        BOOL result;
        GetProperty(0x6, VT_BOOL, (void*)&result);
        return result;
}

void COleFont::SetStrikethrough(BOOL propVal)
{
        SetProperty(0x6, VT_BOOL, propVal);
}

short COleFont::GetWeight()
{
        short result;
        GetProperty(0x7, VT_I2, (void*)&result);
        return result;
}

void COleFont::SetWeight(short propVal)
{
        SetProperty(0x7, VT_I2, propVal);
}

short COleFont::GetCharset()
{
        short result;
        GetProperty(0x8, VT_I2, (void*)&result);
        return result;
}

void COleFont::SetCharset(short propVal)
{
        SetProperty(0x8, VT_I2, propVal);
}

/////////////////////////////////////////////////////////////////////////////
// COleFont operations
```

```
// CntrItem.h : interface of the CThesis16JanCntrItem class
//

#if !defined(AFX_CNTRITEM_H__B4558103_ADE7_11D2_9646_4A728F000000__INCLUDED_)
#define AFX_CNTRITEM_H__B4558103_ADE7_11D2_9646_4A728F000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CThesis16JanDoc;
class CThesis16JanView;

class CThesis16JanCntrItem : public COleClientItem
{
	DECLARE_SERIAL(CThesis16JanCntrItem)

// Constructors
public:
	CThesis16JanCntrItem(CThesis16JanDoc* pContainer = NULL);
		// Note: pContainer is allowed to be NULL to enable IMPLEMENT_SERIALIZE.
		//  IMPLEMENT_SERIALIZE requires the class have a constructor with
		//  zero arguments.  Normally, OLE items are constructed with a
		//  non-NULL document pointer.

// Attributes
public:
	CThesis16JanDoc* GetDocument()
		{ return (CThesis16JanDoc*)COleClientItem::GetDocument(); }
	CThesis16JanView* GetActiveView()
		{ return (CThesis16JanView*)COleClientItem::GetActiveView(); }

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CThesis16JanCntrItem)
	public:
	virtual void OnChange(OLE_NOTIFICATION wNotification, DWORD dwParam);
	virtual void OnActivate();
	protected:
	virtual void OnGetItemPosition(CRect& rPosition);
	virtual void OnDeactivateUI(BOOL bUndoable);
	virtual BOOL OnChangeItemPosition(const CRect& rectPos);
	//}}AFX_VIRTUAL

// Implementation
public:
	~CThesis16JanCntrItem();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
	virtual void Serialize(CArchive& ar);
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_CNTRITEM_H__B4558103_ADE7_11D2_9646_4A728F000000__INCLUDED_)
```

```
// CntrItem.cpp : implementation of the CThesis16JanCntrItem class

//


#include "stdafx.h"

#include "Thesis16Jan.h"


#include "Thesis16JanDoc.h"

#include "Thesis16JanView.h"

#include "CntrItem.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanCntrItem implementation

IMPLEMENT_SERIAL(CThesis16JanCntrItem, COleClientItem, 0)

CThesis16JanCntrItem::CThesis16JanCntrItem(CThesis16JanDoc* pContainer)
        : COleClientItem(pContainer)
{
        // TODO: add one-time construction code here

}

CThesis16JanCntrItem::~CThesis16JanCntrItem()
{
        // TODO: add cleanup code here

}

void CThesis16JanCntrItem::OnChange(OLE_NOTIFICATION nCode, DWORD dwParam)
{
        ASSERT_VALID(this);

        COleClientItem::OnChange(nCode, dwParam);

        // When an item is being edited (either in-place or fully open)
        //  it sends OnChange notifications for changes in the state of the
        //  item or visual appearance of its content.

        // TODO: invalidate the item by calling UpdateAllViews
        //  (with hints appropriate to your application)

        GetDocument()->UpdateAllViews(NULL);
                // for now just update ALL views/no hints
}

BOOL CThesis16JanCntrItem::OnChangeItemPosition(const CRect& rectPos)
{
        ASSERT_VALID(this);
```

```
        // During in-place activation CThesis16JanCntrItem::OnChangeItemPosition
        //  is called by the server to change the position of the in-place
        //  window.  Usually, this is a result of the data in the server
        //  document changing such that the extent has changed or as a result
        //  of in-place resizing.
        //
        // The default here is to call the base class, which will call
        //  COleClientItem::SetItemRects to move the item
        //  to the new position.

        if (!COleClientItem::OnChangeItemPosition(rectPos))
                return FALSE;

        // TODO: update any cache you may have of the item's rectangle/extent

        return TRUE;
}

void CThesis16JanCntrItem::OnGetItemPosition(CRect& rPosition)
{
        ASSERT_VALID(this);

        // During in-place activation, CThesis16JanCntrItem::OnGetItemPosition
        //  will be called to determine the location of this item.  The default
        //  implementation created from AppWizard simply returns a hard-coded
        //  rectangle.  Usually, this rectangle would reflect the current
        //  position of the item relative to the view used for activation.
        //  You can obtain the view by calling CThesis16JanCntrItem::GetActiveView.

        // TODO: return correct rectangle (in pixels) in rPosition

        rPosition.SetRect(10, 10, 210, 210);
}

void CThesis16JanCntrItem::OnActivate()
{
    // Allow only one inplace activate item per frame
    CThesis16JanView* pView = GetActiveView();
    ASSERT_VALID(pView);
    COleClientItem* pItem = GetDocument()->GetInPlaceActiveItem(pView);
    if (pItem != NULL && pItem != this)
        pItem->Close();

    COleClientItem::OnActivate();
}

void CThesis16JanCntrItem::OnDeactivateUI(BOOL bUndoable)
{
        COleClientItem::OnDeactivateUI(bUndoable);

    // Hide the object if it is not an outside-in object
    DWORD dwMisc = 0;
    m_lpObject->GetMiscStatus(GetDrawAspect(), &dwMisc);
    if (dwMisc & OLEMISC_INSIDEOUT)
        DoVerb(OLEIVERB_HIDE, NULL);
}

void CThesis16JanCntrItem::Serialize(CArchive& ar)
{
        ASSERT_VALID(this);

        // Call base class first to read in COleClientItem data.
        // Since this sets up the m_pDocument pointer returned from
```

```
        //  CThesis16JanCntrItem::GetDocument, it is a good idea to call
        //  the base class Serialize first.
        COleClientItem::Serialize(ar);

        // now store/retrieve data specific to CThesis16JanCntrItem
        if (ar.IsStoring())
        {
                // TODO: add storing code here
        }
        else
        {
                // TODO: add loading code here
        }
}

/////////////////////////////////////////////////////////////////////////////
// CThesis16JanCntrItem diagnostics

#ifdef _DEBUG
void CThesis16JanCntrItem::AssertValid() const
{
        COleClientItem::AssertValid();
}

void CThesis16JanCntrItem::Dump(CDumpContext& dc) const
{
        COleClientItem::Dump(dc);
}
#endif

/////////////////////////////////////////////////////////////////////////////
```

```
// Float.h: interface for the CFloat class.

//

//////////////////////////////////////////////////////////////////////

#if !defined(AFX_FLOAT_H__5BF87D8C_BD60_11D2_9682_B0FD91000000__INCLUDED_)
#define AFX_FLOAT_H__5BF87D8C_BD60_11D2_9682_B0FD91000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define LATITUDE 0
#define LONGITUDE 1
#define ALTITUDE 2
#define HEADING 3
#define SPEED 4

class CFloat
{
public:
        bool SetType(int);
        CString GetCheckSum();
        bool GetSign();
        CString GetString();
        float GetFloat();
        CFloat();
        virtual ~CFloat();

        bool operator= (float);

private:
        bool SetCheckSum(CString);
        CString csCheckSum;
        int f_Type;
        bool bSign;
        int iFraction;
        int iInteger;
        CString csValue;
        float fValue;


};

#endif // !defined(AFX_FLOAT_H__5BF87D8C_BD60_11D2_9682_B0FD91000000__INCLUDED_)
```

```
// Float.cpp: implementation of the CFloat class.

//

//////////////////////////////////////////////////////////////////////



#include "stdafx.h"

#include "Thesis16Jan.h"

#include "Float.h"

#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFloat::CFloat()
{
        bSign = false;
        fValue = 0.0F;
        iInteger = 0;
        iFraction = 0;
        csValue = "";
        f_Type = 0;     // I'm not using this...
        csCheckSum = "";
}

CFloat::~CFloat()
{

}

bool CFloat::operator = (float fArgv)
{
        unsigned int temp;

        fValue = fArgv;
        if (fArgv < 0) bSign = true;
        else bSign = false;
        fArgv = (float)fabs(fArgv);
        iInteger = (int)fArgv;
        switch (f_Type)
        {
        case 0: // latitude
                {
                iFraction = (int)((fArgv - (float)iInteger) * 60);
                temp = iInteger * 100 + iFraction;
                csValue.Format("%04d", temp);
                SetCheckSum(csValue);
                if (bSign)
                {
                        csValue += "S";
                }
                else
```

```
                {
                        csValue += "N";
                }
                csValue += csCheckSum;
                break;
                }
        case 1: // longitude
                {
                iFraction = (int)((fArgv - (float)iInteger) * 60);
                temp = iInteger * 100 + iFraction;
                csValue.Format("%05d", temp);
                SetCheckSum(csValue);
                if (bSign)
                {
                        csValue += "W";
                }
                else
                {
                        csValue += "E";
                }
                csValue += csCheckSum;
                break;
                }
        case 2: // altitude
                {
                iFraction = (int)((fArgv - (float)iInteger));
                temp = iInteger / 100;
                csValue.Format("%03d", temp);
                csValue = "ALT" + csValue;
                break;
                }
        case 3: // heading
                {
                iFraction = (int)((fArgv - (float)iInteger));
                temp = iInteger;
                csValue.Format("%03d", temp);
                csValue += "T";
                break;
                }
        case 4: // speed
                {
                iFraction = (int)((fArgv - (float)iInteger) * 100);
                temp = iInteger;
                csValue.Format("%d", temp);
                csValue += "K";
                break;
                }
        default: // all others
                {
                temp = iInteger;
                csValue.Format("%d", temp);
                break;
                }
        }

        return true;
}

float CFloat::GetFloat()
{
        return fValue;
}
```

```
CString CFloat::GetString()
{
	return (csValue);
}

bool CFloat::GetSign()
{
	return bSign;
}

CString CFloat::GetCheckSum()
{
	return csCheckSum;
}

bool CFloat::SetType(int iType)
{
	f_Type = iType;
	return true;
}

bool CFloat::SetCheckSum(CString csValue)
{

	int iCheckSum = 0;
	int ptr;

	for (ptr = 0; ptr < csValue.GetLength(); ptr++)
	{
		iCheckSum += atoi(csValue.Mid(ptr,1));
	}
	csCheckSum.Format("%d", iCheckSum);
	csCheckSum = csCheckSum.Right(1);
	return true;
}
```

```
// CSerial.h

// This class add serial connectivity to the project!
//

class CSerial
{


public:


private:

        struct sTelemetry
        {
                float fLatitude;
                float fLongitude;
                float fAltitude;
                float fTimeOfFix;
                float fPitch;
                float fRoll;
                float fHeading;
                float fGLatitude;
                float fGLongitude;
                float fGAltitude;
                float fGTimeOfFix;
                float fVGPitch;
                float fVGRoll;
                float fMHeading;
                float fSpeed;
        };

        union floatConvert
        {
                byte floatBuffer[60];
                sTelemetry telemetryData;
        };

        struct sFloat
        {
                bool bSign;
                CString CSFloat;
                CString CSCheckSum;
        };

        float fLatitude;
        float fLongitude;
        float fAltitude;
        float fHeading;
        float fSpeed;
        float fTime;

}
```

```
// CTelemetry.h

#include "Float.h"     // Added by ClassView

#include "Structures.h"


class CTelemetry
{
public:
        bool SetCommPort(int);

        CTelemetry();
        bool OpenCommPort();
        bool ReadTelemetry();
        //bool ProcessTelemetry();
        bool ProcessTelemetry(StrCSTm*);
        bool CloseCommPort();

private:
        CString csMonth;
        CString GetMonth();
        CString csComPort;
        CString GetTime();

        CFloat cfLatitude;
        CFloat cfLongitude;
        CFloat cfAltitude;
        CFloat cfHeading;
        CFloat cfSpeed;

        byte holdData;
        int TestByte (byte bTest);

        byte readData[256];
        HANDLE hComm;
        DCB dcbComm;

        int days;
        int hours;
        int minutes;

        DWORD dwNrBytesRead;

        StrFltTm SFltTm; // Structure containing float TM
};
```

```
// CTelemetry.cpp

#include "stdafx.h"
#include "resource.h"
#include "CTelemetry.h"
#include "Float.h"
//#include "Structures.h"

CTelemetry::CTelemetry()
{
        CTime ctTime;
        ctTime = CTime::GetCurrentTime();

        days = ctTime.GetDay();
        hours = ctTime.GetHour();
        minutes = ctTime.GetMinute();

        cfLatitude.SetType(LATITUDE);
        cfLongitude.SetType(LONGITUDE);
        cfAltitude.SetType(ALTITUDE);
        cfHeading.SetType(HEADING);
        cfSpeed.SetType(SPEED);
}

bool CTelemetry::ReadTelemetry()
{
        dwNrBytesRead = 0;

        ReadFile(hComm,
                        readData,
                        sizeof(readData),
                        &dwNrBytesRead,
                        NULL);
        if (dwNrBytesRead == 0)
        {
                return false;
        }
        return true;
}

bool CTelemetry::OpenCommPort()
{
        COMMTIMEOUTS commTo = {0};
        hComm = CreateFile(csComPort,
                                        GENERIC_READ,
                                        0,
                                        NULL,
                                        OPEN_EXISTING,
                                        NULL,
                                        NULL);
        if (hComm == INVALID_HANDLE_VALUE)
        {
                AfxMessageBox ("Communications Error on COM2");
                return FALSE;
        }
        else
        {
                if (!GetCommState(hComm, &dcbComm))
                {
                        return FALSE;
                }
                dcbComm.BaudRate = CBR_19200;
                dcbComm.fBinary = TRUE;
                dcbComm.ByteSize = 8;
```

```
                dcbComm.Parity = NOPARITY;
                dcbComm.StopBits = ONESTOPBIT;
                SetCommState(hComm, &dcbComm);
        }

        commTo.ReadIntervalTimeout = 1;
        commTo.ReadTotalTimeoutConstant = 1;
        commTo.ReadTotalTimeoutMultiplier = 1;
        if (!SetCommTimeouts(hComm, &commTo))
        {
                AfxMessageBox("Error setting timeouts!");
                return false;
        }
        return true;
}

bool CTelemetry::CloseCommPort()
{
        CloseHandle(hComm);
        return true;
}

bool CTelemetry::ProcessTelemetry(StrCSTm *SCSTm)

{
        union
        {
                byte bBuffer[60];
                StrRawTm tData;
        } uBuffer = {0};

        const int FRAMESTART = -1;
        const int FRAMESTUFFING = 0;
        const int FRAMEEND = 1;
        const int FRAMEERROR = 2;
        int x=0;
        byte nextCharFlag;
        bool fProcessing = TRUE;
        bool fFrameStart = FALSE;
        bool fFrameEnd = FALSE;

        // Look for FRAMESTART
        while ((!fFrameStart) && (dwNrBytesRead != 0))
        {
                if (readData[x] == 0x10)
                {
                        if (TestByte(readData[x+1])==FRAMESTART)
                        {
                                // OK.  Frame found.  Continue processing
                                fFrameStart = TRUE;
                                x+=2;
                        }
                        else
                        {
                                // Frame not found, bail out
                                return false;
                        }
                }
                else
                {
                        x+=1;
                }
        }
```

```
int bufPointer = 0;

// Unstuff the received data an copy to a union.
while (!fFrameEnd)
{
        holdData = readData[x];
        if (holdData == 0x10)
        {
                nextCharFlag = TestByte(readData[x+1]);
                switch (nextCharFlag)
                {
                case FRAMESTART:        // Bad framing return to calling function
                        {                               // bail out
                                return false;
                        }
                case FRAMESTUFFING: // Stuffing detected, adjust data pointer
                        {
                                x+=1;
                                break;
                        }
                case FRAMEEND:          // End of frame
                        {                               // Done processing frame
                                fFrameEnd = true;
                                break;
                        }
                case FRAMEERROR:        // Severe data error
                        {                               // bail out
                                AfxMessageBox ("Unknown error in telemetry data");
                                return false;
                        }
                }
        }
        // Check for end of frame and buffer pointer constraints before
        // moving data
        if ((!fFrameEnd) && (bufPointer < 60))
        {
                uBuffer.bBuffer[bufPointer] = holdData;
                bufPointer += 1;
        }
        x++;
        // Bail out if data buffer is really screwed up.
        if (x>255) return false;
}
//
//      Got the floating point values, now convert them to a
//      CFloat
//
cfLatitude = uBuffer.tData.fltLatitude * float(180/3.1415) ;
cfLongitude = uBuffer.tData.fltLongitude * float(180/3.1415);
cfAltitude = uBuffer.tData.fltAltitude * float(3.28);
cfHeading = uBuffer.tData.fltHeading;
cfSpeed = uBuffer.tData.fltSpeed;
//
//      Load a StrCSTm structure for further processing
//
SCSTm->csLatitude = cfLatitude.GetString();
SCSTm->csLongitude = cfLongitude.GetString();
SCSTm->csAltitude = cfAltitude.GetString();
SCSTm->csHeading = cfHeading.GetString();
SCSTm->csSpeed = cfSpeed.GetString();
SCSTm->csTime = GetTime();
SCSTm->csMonth = GetMonth();

return true;
```

```
}

int CTelemetry::TestByte (byte bTest)
{
        switch (bTest)
        {
                case 0x3:
                {
                        return 1;
                        break;
                }
                case 0x10:
                {
                        return 0;
                        break;
                }
                default :
                {
                        return -1;
                        break;
                }
        }
        return 2;
}

CString CTelemetry::GetTime()
{
        CTime ctTime;
        CString retValue;
        CString csCheckSum;

        int iCheckSum = 0;
        int ptr;

        ctTime = CTime::GetCurrentTime();
        csMonth = ctTime.FormatGmt("%b");
        csMonth.MakeUpper();
        retValue = ctTime.FormatGmt("%d%H%M");

        //days = ctTime.GetDay();
        //hours = ctTime.GetHour();
        //minutes = ctTime.GetMinute();

        //iTime = 10000 * days + 100 * hours + minutes;
        //retValue.Format("%06d", iTime);

        for (ptr = 0; ptr < retValue.GetLength(); ptr++)
        {
                iCheckSum += atoi(retValue.Mid(ptr, 1));
        }
        csCheckSum.Format("%d", iCheckSum);
        csCheckSum = csCheckSum.Right(1);

        retValue = retValue + "Z" + csCheckSum;
        return retValue;
}

bool CTelemetry::SetCommPort(int inValue)
{
        switch (inValue)
        {
        case 0:
                {
                        csComPort = "COM1";
```

```
                        break;
                }
        case 1:
                {
                        csComPort = "COM2";
                        break;
                }
        case 2:
                {
                        csComPort = "COM3";
                        break;
                }
        case 3:
                {
                        csComPort = "COM4";
                        break;
                }
        default:
                {
                        csComPort = "COM1";
                        break;
                }
        }
        return true;
}

CString CTelemetry::GetMonth()
{
        return csMonth;
}
```

```
//  OTH-T Gold
//      CMsgGold
//      CMsgGold.h

#if !defined CMSGGOLD
#define CMSGGOLD

#include "Structures.h"
class CMsgGold
{
public:

        CMsgGold();

        bool LoadData(StrCSTm*);
        bool SetCommPort(int);

        // Load static message strings
        bool LoadString(int);
        // Load dynamic data
        CString LoadData(CString, CString, CString, CString);
        // Build the OTH-T Gold
        CString MakeMsgGold();
        CString CSRelay;

private:

        int iSerial;
        int iTrack;

        DWORD dwLength;

        CString csComPort;
        CString csMessage;
        CString csTrack;
        CString csBt;
        CString csMsgid;
        CString csCtc;
        CString csEndat;
        CString csPos;
        CString csSerial;
        CString csMonth;
        CString csLatitude;
        CString csLongitude;
        CString csAltitude;
        CString csHeading;
        CString csSpeed;
        CString csTime;
        CString csAltusTrack;

        // Send message via serial port
        bool SendMsgGold(CString);
        // Send message via TCP/IP
        bool SendMsgGoldIP(CString);
        // Read message via TCP/IP
        bool ReadMsgGoldIP();
};

#endif
```

```
//  OTH-T Gold
//      CMsgGold
//      CMsgGold.cpp

#include "stdafx.h"
#include "resource.h"
#include "CMsgGold.h"
#include "DlgHostname.h"

bool CMsgGold::LoadString(int iType)
{
        csBt.LoadString(IDS_OTH_GOLD_BT);
        csMsgid.LoadString(IDS_OTH_GOLD_MSGID);
        csPos.LoadString(IDS_OTH_GOLD_POS);
        csEndat.LoadString(IDS_OTH_GOLD_ENDAT);

        switch(iType)
        {
        case 1:
                {
                csCtc.LoadString(IDS_OTH_GOLD_CTC);
                break;
                }
        case 2:
                {
                csCtc.LoadString(IDS_OTH_GOLD_CTC_REPORT);
                break;
                }
        }

        return true;
}

CString CMsgGold::MakeMsgGold()
{
        csMessage = csBt + "\r\n";
        csMessage += csMsgid  + csSerial + "/" + csMonth + "/" + "\r\n";
        csMessage += csCtc + "\r\n";
        csMessage += csPos + csTime + "/" + csMonth + "/" + csLatitude
                                + "/" + csLongitude + "/////" + csHeading + "/"
                                + csSpeed + "/" + csAltitude + "\r\n";
        csMessage += csEndat + "\r\n";
        csMessage += csBt + "\r\n";

        dwLength = csMessage.GetLength();
        SendMsgGold(csMessage);
        return csMessage;
}

bool CMsgGold::SendMsgGold(CString csMsg)
{
        if (csComPort != "TCP")
        {
                DCB dcbComm1;
                HANDLE hComm1;
                DWORD nrBytesWritten = 0;

                hComm1 = CreateFile(csComPort,
                                        GENERIC_READ | GENERIC_WRITE,
                                        0,
                                        NULL,
                                        OPEN_EXISTING,
                                        NULL,
                                        NULL);
```

```
                if (hComm1 == INVALID_HANDLE_VALUE)
                {
                        AfxMessageBox("Error opening " + csComPort);
                        return false;
                }
                if (!GetCommState(hComm1, &dcbComm1))
                {
                        AfxMessageBox("Error in GetCommState");
                        return false;
                }
                dcbComm1.BaudRate = CBR_19200;
                dcbComm1.fBinary = TRUE;
                dcbComm1.ByteSize = 8;
                dcbComm1.Parity = NOPARITY;
                dcbComm1.StopBits = ONESTOPBIT;
                dcbComm1.fDtrControl = DTR_CONTROL_DISABLE;
                dcbComm1.fOutX = FALSE;
                dcbComm1.fInX = FALSE;
                dcbComm1.fRtsControl = RTS_CONTROL_DISABLE;

                if (!SetCommState(hComm1, &dcbComm1))
                {
                        AfxMessageBox("Error setting com state on " + csComPort);
                        return false;
                }

                DWORD dwNrBytesToWrite;
                dwNrBytesToWrite = dwLength;

                if (!WriteFile(hComm1,
                                csMsg,
                                dwNrBytesToWrite,
                                &nrBytesWritten,
                                NULL))
                {
                        AfxMessageBox("Error writing to writing to " + csComPort);
                        return false;
                }
                CloseHandle(hComm1);
        }
        else
        {
                CSocket gccsSocket;
                gccsSocket.Create();
                if (CSRelay.IsEmpty())
                {
                        CDlgHostname dlgHostName;
                        dlgHostName.DoModal(&CSRelay);
                }
                gccsSocket.Connect(CSRelay,2071);
                gccsSocket.Send(csMsg, csMsg.GetLength());
                gccsSocket.Close();
        }
        return true;
}

bool ReadMsgGoldIP()
{
        // This member function is used to listen to an IP Port for
        // an OTH-T Gold message, the message should be passed to a
        // Gateway using SendMsgGoldSer(CString)
        return true;
}
        //bool LoadString();
```

```
        //bool SendMsgGoldIP();
        //bool SendMsgGoldSer();


bool CMsgGold::SetCommPort(int inValue)
{
        switch (inValue)
        {
        case 0:
                {
                        csComPort = "COM1";
                        break;
                }
        case 1:
                {
                        csComPort = "COM2";
                        break;
                }
        case 2:
                {
                        csComPort = "COM3";
                        break;
                }
        case 3:
                {
                        csComPort = "COM4";
                        break;
                }
        case 4:
                {
                        csComPort = "TCP";
                        break;
                }
        default:
                {
                        csComPort = "COM1";
                        break;
                }
        }
        return true;
}

bool CMsgGold::LoadData(StrCSTm* strIn)
{
        csSerial.Format("%04d", iSerial++);
        csLatitude = strIn->csLatitude;
        csLongitude = strIn->csLongitude;
        csHeading = strIn->csHeading;
        csSpeed = strIn->csSpeed;
        csAltitude = strIn->csAltitude;
        csMonth = strIn->csMonth;
        csTime = strIn->csTime;
        return true;


}

CString CMsgGold::LoadData(CString vLatitude,
                                                CString vLongitude,
                                                CString vTime,
                                                CString vMonth)
{
        csSerial.Format("%04d", iSerial++);
        csLatitude = vLatitude;
```

```
        csLongitude = vLongitude;
        csTrack.Format("%05d", iTrack++);
        csTrack = "T" + csTrack;
        csHeading = "";
        csSpeed = "";
        csAltitude = "";
        csMonth = vMonth;
        csTime = vTime;
        csCtc = "CTC/" + csTrack + "/UNEQUATED-UNKNOWN/";
        return csTrack;
}

CMsgGold::CMsgGold()
{
        iSerial = 1;
        iTrack = 2;
        csAltusTrack = "T0001";
        CSRelay = "";
}
```

```
#if !defined(AFX_DIALOGSETTINGS_H__6109EEB7_BDF5_11D2_9683_38F3A7000000__INCLUDED_)
#define AFX_DIALOGSETTINGS_H__6109EEB7_BDF5_11D2_9683_38F3A7000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DialogSettings.h : header file
//
#include "Structures.h"
/////////////////////////////////////////////////////////////////////////////
// CDialogSettings dialog

class CDialogSettings : public CDialog
{
// Construction
public:

        int CDialogSettings::DoModal(stIniTxt* InArgv);
        //int DoModal(CString* pcsInArgv);
        CDialogSettings(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CDialogSettings)
        enum { IDD = IDD_DLG_SETTINGS };
        int             m_Telemetry;
        int             m_Gccs;
        int             m_Board;

        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDialogSettings)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation


protected:

        // Generated message map functions
        //{{AFX_MSG(CDialogSettings)
        virtual BOOL OnInitDialog();
        virtual void OnOK();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
        CString m_csGateway;
        CString m_csDomain;
        CString m_csImagePath;

        stIniTxt* stIniStrings;
        CWnd* cParent;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_DIALOGSETTINGS_H__6109EEB7_BDF5_11D2_9683_38F3A7000000__INCLUDED_)
```

```
// DialogSettings.cpp : implementation file
//

#include "stdafx.h"
#include "Thesis16Jan.h"
#include "DialogSettings.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDialogSettings dialog

CDialogSettings::CDialogSettings(CWnd* pParent /*=NULL*/)
        : CDialog(CDialogSettings::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDialogSettings)
        m_Telemetry = -1;
        m_Gccs = -1;
        m_Board = -1;
        m_csGateway = _T("");
        //}}AFX_DATA_INIT
}


void CDialogSettings::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDialogSettings)
        DDX_Radio(pDX, IDC_RD_COM_A1, m_Telemetry);
        DDX_Radio(pDX, IDC_RD_COM_B1, m_Gccs);
        DDX_Radio(pDX, IDC_RD_BOARD1, m_Board);
        DDX_Text(pDX, IDC_EDIT_GATEWAY, m_csGateway);
        DDX_Text(pDX, IDC_EDIT_DOMAIN, m_csDomain);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDialogSettings, CDialog)
        //{{AFX_MSG_MAP(CDialogSettings)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDialogSettings message handlers

BOOL CDialogSettings::OnInitDialog()
{
        CString m_csGccs;
        CString m_csTelemetry;
        CString m_csBoard;


        CStdioFile iniFile;
        iniFile.Open(".\\altus.ini", CFile::modeRead);
        iniFile.ReadString(m_csGateway);
        iniFile.ReadString(m_csDomain);
        iniFile.ReadString(m_csBoard);
        iniFile.ReadString(m_csTelemetry);
        iniFile.ReadString(m_csGccs);
        iniFile.ReadString(m_csImagePath);
        iniFile.Close();
```

```
        m_Gccs = atoi(m_csGccs.Left(1));
        m_Telemetry = atoi(m_csTelemetry.Left(1));
        m_Board = atoi(m_csBoard.Left(1));

        UpdateData(false);

        CDialog::OnInitDialog();
        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}

int CDialogSettings::DoModal(stIniTxt* InArgv)
//int CDialogSettings::DoModal(CString* pcsInArgv)
{
        stIniStrings = InArgv;
        int retValue;

        CDialog::DoModal();
        retValue = 256 * m_Board + 16 * m_Telemetry + m_Gccs;
        return retValue;
}

void CDialogSettings::OnOK()
{
        UpdateData(true);
        stIniStrings->csFQDN.Empty();
        stIniStrings->csImagePath.Empty();

        stIniStrings->csFQDN.Insert(0, m_csGateway + "." + m_csDomain);
        stIniStrings->csImagePath.Insert(0, m_csImagePath);

        CDialog::OnOK();
}
```

```
#if !defined(AFX_DLGHOSTNAME_H__43386FF2_BF40_11D2_9685_725961000000__INCLUDED_)
#define AFX_DLGHOSTNAME_H__43386FF2_BF40_11D2_9685_725961000000__INCLUDED_
#include "CMsgGold.h"
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DlgHostname.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDlgHostname dialog

class CDlgHostname : public CDialog
{
// Construction
public:
        int DoModal(CString* csInArgv);
        CDlgHostname(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CDlgHostname)
        enum { IDD = IDD_DLG_HOSTNAME };
        CString m_relay;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDlgHostname)
        public:

        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CDlgHostname)
        virtual BOOL OnInitDialog();
        virtual void OnOK();
        afx_msg void OnKillfocusEditHostname();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
private:
        CString* pCSRelay;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_DLGHOSTNAME_H__43386FF2_BF40_11D2_9685_725961000000__INCLUDED_)
```

```
// DlgHostname.cpp : implementation file
//

#include "stdafx.h"
#include "Thesis16Jan.h"
#include "DlgHostname.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDlgHostname dialog


CDlgHostname::CDlgHostname(CWnd* pParent /*=NULL*/)
        : CDialog(CDlgHostname::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDlgHostname)
        m_relay = _T("");
        //}}AFX_DATA_INIT
}


void CDlgHostname::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDlgHostname)
        DDX_Text(pDX, IDC_EDIT_HOSTNAME, m_relay);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CDlgHostname, CDialog)
        //{{AFX_MSG_MAP(CDlgHostname)
        ON_EN_KILLFOCUS(IDC_EDIT_HOSTNAME, OnKillfocusEditHostname)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDlgHostname message handlers

int CDlgHostname::DoModal(CString* csInArgv)
{
        pCSRelay = csInArgv;
        pCSRelay->Empty();
        CDialog::DoModal();
        return 0;
}

BOOL CDlgHostname::OnInitDialog()
{
        CDialog::OnInitDialog();
        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}

void CDlgHostname::OnOK()
{

        CDialog::OnOK();
}
```

```
void CDlgHostname::OnKillfocusEditHostname()
{
        UpdateData(true);
        pCSRelay->Insert(0, m_relay);
}
```

```
#if !defined(AFX_CAPTURE_H__B455810B_ADE7_11D2_9646_4A728F000000__INCLUDED_)
#define AFX_CAPTURE_H__B455810B_ADE7_11D2_9646_4A728F000000__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// Machine generated IDispatch wrapper class(es) created by Microsoft Visual C++

// NOTE: Do not modify the contents of this file.  If this class is regenerated by
//  Microsoft Visual C++, your modifications will be overwritten.


// Dispatch interfaces referenced by this interface
class COleFont;
class CPicture;

/////////////////////////////////////////////////////////////////////////////
// CCapture wrapper class

class CCapture : public CWnd
{
protected:
        DECLARE_DYNCREATE(CCapture)
public:
        CLSID const& GetClsid()
        {
                static CLSID const clsid
                = {0xcc34cebf, 0x5c10, 0x11d1, { 0xa4, 0xf, 0x0, 0xa0, 0x24, 0x22, 0x9c,
0x83 } };
                return clsid;
        }
        virtual BOOL Create(LPCTSTR lpszClassName,
                LPCTSTR lpszWindowName, DWORD dwStyle,
                const RECT& rect,
                CWnd* pParentWnd, UINT nID,
                CCreateContext* pContext = NULL)
        { return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect, pParentWnd,
nID); }

    BOOL Create(LPCTSTR lpszWindowName, DWORD dwStyle,
                const RECT& rect, CWnd* pParentWnd, UINT nID,
                CFile* pPersist = NULL, BOOL bStorage = FALSE,
                BSTR bstrLicKey = NULL)
        { return CreateControl(GetClsid(), lpszWindowName, dwStyle, rect, pParentWnd, nID,
                pPersist, bStorage, bstrLicKey); }

// Attributes
public:

// Operations
public:
        void SetRefFont(LPDISPATCH newValue);
        void SetFont(LPDISPATCH newValue);
        COleFont GetFont();
        void SetCaption(LPCTSTR lpszNewValue);
        CString GetCaption();
        void SetBorderVisible(BOOL bNewValue);
        BOOL GetBorderVisible();
        void SetBorderWidth(long nNewValue);
        long GetBorderWidth();
        void SetEnabled(BOOL bNewValue);
        BOOL GetEnabled();
        long GetWindow();
        CPicture GetPicture();
```

```
void SetForeColor(unsigned long newValue);
unsigned long GetForeColor();
void About();
void Connect(long DeviceIndex);
long GetDeviceIndex();
CString GetDeviceName(long Index);
CString GetDeviceVersion(long Index);
BOOL GetHasOverlay();
BOOL GetHasDlgVideoSource();
BOOL GetHasDlgVideoFormat();
BOOL GetHasDlgVideoDisplay();
BOOL GetDriverSuppliesPalettes();
BOOL GetPreview();
void SetPreview(BOOL bNewValue);
void Disconnect();
void ShowVideoFormatDlg();
void ShowVideoSourceDlg();
void ShowVideoDisplayDlg();
void ShowVideoCompressionDlg();
CString GetFrameFile();
void SetFrameFile(LPCTSTR lpszNewValue);
void CaptureFrame();
long GetNumDevices();
long GetSaveJPGChromFactor();
void SetSaveJPGChromFactor(long nNewValue);
BOOL GetSaveJPGProgressive();
void SetSaveJPGProgressive(BOOL bNewValue);
long GetSaveJPGLumFactor();
void SetSaveJPGLumFactor(long nNewValue);
CString GetPICPassword();
void SetPICPassword(LPCTSTR lpszNewValue);
BOOL GetAutoSave();
void SetAutoSave(BOOL bNewValue);
long GetInterval();
void SetInterval(long nNewValue);
long GetPreviewRate();
void SetPreviewRate(long nNewValue);
long GetCaptionLeft();
void SetCaptionLeft(long nNewValue);
long GetCaptionTop();
void SetCaptionTop(long nNewValue);
long GetCaptionWidth();
void SetCaptionWidth(long nNewValue);
long GetCaptionHeight();
void SetCaptionHeight(long nNewValue);
BOOL GetShadowText();
void SetShadowText(BOOL bNewValue);
BOOL GetClipCaption();
void SetClipCaption(BOOL bNewValue);
CString GetStreamFile();
void SetStreamFile(LPCTSTR lpszNewValue);
void StartCapture();
void EndCapture();
long GetFrameRate();
void SetFrameRate(long nNewValue);
BOOL GetYield();
void SetYield(BOOL bNewValue);
BOOL GetCaptureAudio();
void SetCaptureAudio(BOOL bNewValue);
long GetTimeLimit();
void SetTimeLimit(long nNewValue);
long GetFramesProcessed();
long GetFramesDropped();
long GetWaveSamples();
```

```
        long GetTimeElapsed();
        long GetAudioChannels();
        void SetAudioChannels(long nNewValue);
        long GetAudioBits();
        void SetAudioBits(long nNewValue);
        long GetAudioSampleRate();
        void SetAudioSampleRate(long nNewValue);
        CString GetFTPUserName();
        void SetFTPUserName(LPCTSTR lpszNewValue);
        CString GetFTPPassword();
        void SetFTPPassword(LPCTSTR lpszNewValue);
        long GetHDib();
        long GetSaveJPGSubSampling();
        void SetSaveJPGSubSampling(long nNewValue);
        BOOL GetAutoIncrement();
        void SetAutoIncrement(BOOL bNewValue);
        long GetHAlign();
        void SetHAlign(long nNewValue);
        long GetVAlign();
        void SetVAlign(long nNewValue);
        CString GetProxyServer();
        void SetProxyServer(LPCTSTR lpszNewValue);
        void GrabFrame();
        void SaveFrame();
        BOOL GetOverlay();
        void SetOverlay(BOOL bNewValue);
        BOOL GetFTPRename();
        void SetFTPRename(BOOL bNewValue);
        long GetResX();
        void SetResX(long nNewValue);
        long GetResY();
        void SetResY(long nNewValue);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_CAPTURE_H__B455810B_ADE7_11D2_9646_4A728F000000__INCLUDED_)
```

```
// Machine generated IDispatch wrapper class(es) created by Microsoft Visual C++
// NOTE: Do not modify the contents of this file.  If this class is regenerated by
//  Microsoft Visual C++, your modifications will be overwritten.

#include "stdafx.h"
#include "capture.h"

// Dispatch interfaces referenced by this interface
#include "Font.h"
#include "Picture.h"

/////////////////////////////////////////////////////////////////////////////
// CCapture

IMPLEMENT_DYNCREATE(CCapture, CWnd)

/////////////////////////////////////////////////////////////////////////////
// CCapture properties

/////////////////////////////////////////////////////////////////////////////
// CCapture operations

void CCapture::SetRefFont(LPDISPATCH newValue)
{
        static BYTE parms[] =
                VTS_DISPATCH;
        InvokeHelper(DISPID_FONT, DISPATCH_PROPERTYPUTREF, VT_EMPTY, NULL, parms,
                 newValue);
}

void CCapture::SetFont(LPDISPATCH newValue)
{
        static BYTE parms[] =
                VTS_DISPATCH;
        InvokeHelper(DISPID_FONT, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 newValue);
}

COleFont CCapture::GetFont()
{
        LPDISPATCH pDispatch;
        InvokeHelper(DISPID_FONT, DISPATCH_PROPERTYGET, VT_DISPATCH, (void*)&pDispatch,
NULL);
        return COleFont(pDispatch);
}

void CCapture::SetCaption(LPCTSTR lpszNewValue)
{
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(DISPID_CAPTION, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 lpszNewValue);
}

CString CCapture::GetCaption()
{
        CString result;
        InvokeHelper(DISPID_CAPTION, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, NULL);
        return result;
}

void CCapture::SetBorderVisible(BOOL bNewValue)
{
        static BYTE parms[] =
```

```
            VTS_BOOL;
        InvokeHelper(0xfffffdf9, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                bNewValue);
}

BOOL CCapture::GetBorderVisible()
{
        BOOL result;
        InvokeHelper(0xfffffdf9, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

void CCapture::SetBorderWidth(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0xfffffe07, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                nNewValue);
}

long CCapture::GetBorderWidth()
{
        long result;
        InvokeHelper(0xfffffe07, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetEnabled(BOOL bNewValue)
{
        static BYTE parms[] =
                VTS_BOOL;
        InvokeHelper(DISPID_ENABLED, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                bNewValue);
}

BOOL CCapture::GetEnabled()
{
        BOOL result;
        InvokeHelper(DISPID_ENABLED, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

long CCapture::GetWindow()
{
        long result;
        InvokeHelper(DISPID_HWND, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

CPicture CCapture::GetPicture()
{
        LPDISPATCH pDispatch;
        InvokeHelper(0xfffffdf5, DISPATCH_PROPERTYGET, VT_DISPATCH, (void*)&pDispatch,
NULL);
        return CPicture(pDispatch);
}

void CCapture::SetForeColor(unsigned long newValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(DISPID_FORECOLOR, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                newValue);
}
```

```
unsigned long CCapture::GetForeColor()
{
        unsigned long result;
        InvokeHelper(DISPID_FORECOLOR, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::About()
{
        InvokeHelper(0xfffffdd8, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CCapture::Connect(long DeviceIndex)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x1, DISPATCH_METHOD, VT_EMPTY, NULL, parms,
                 DeviceIndex);
}

long CCapture::GetDeviceIndex()
{
        long result;
        InvokeHelper(0x2, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

CString CCapture::GetDeviceName(long Index)
{
        CString result;
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x3, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, parms,
                Index);
        return result;
}

CString CCapture::GetDeviceVersion(long Index)
{
        CString result;
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x4, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, parms,
                Index);
        return result;
}

BOOL CCapture::GetHasOverlay()
{
        BOOL result;
        InvokeHelper(0x5, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

BOOL CCapture::GetHasDlgVideoSource()
{
        BOOL result;
        InvokeHelper(0x6, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

BOOL CCapture::GetHasDlgVideoFormat()
{
```

```
        BOOL result;
        InvokeHelper(0x7, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

BOOL CCapture::GetHasDlgVideoDisplay()
{
        BOOL result;
        InvokeHelper(0x8, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

BOOL CCapture::GetDriverSuppliesPalettes()
{
        BOOL result;
        InvokeHelper(0x9, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

BOOL CCapture::GetPreview()
{
        BOOL result;
        InvokeHelper(0xa, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

void CCapture::SetPreview(BOOL bNewValue)
{
        static BYTE parms[] =
                VTS_BOOL;
        InvokeHelper(0xa, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 bNewValue);
}

void CCapture::Disconnect()
{
        InvokeHelper(0xb, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CCapture::ShowVideoFormatDlg()
{
        InvokeHelper(0xc, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CCapture::ShowVideoSourceDlg()
{
        InvokeHelper(0xd, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CCapture::ShowVideoDisplayDlg()
{
        InvokeHelper(0xe, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CCapture::ShowVideoCompressionDlg()
{
        InvokeHelper(0xf, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

CString CCapture::GetFrameFile()
{
        CString result;
        InvokeHelper(0x10, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, NULL);
        return result;
```

```
}

void CCapture::SetFrameFile(LPCTSTR lpszNewValue)
{
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x10, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                lpszNewValue);
}

void CCapture::CaptureFrame()
{
        InvokeHelper(0x11, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

long CCapture::GetNumDevices()
{
        long result;
        InvokeHelper(0x12, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

long CCapture::GetSaveJPGChromFactor()
{
        long result;
        InvokeHelper(0x13, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetSaveJPGChromFactor(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x13, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                nNewValue);
}

BOOL CCapture::GetSaveJPGProgressive()
{
        BOOL result;
        InvokeHelper(0x14, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

void CCapture::SetSaveJPGProgressive(BOOL bNewValue)
{
        static BYTE parms[] =
                VTS_BOOL;
        InvokeHelper(0x14, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                bNewValue);
}

long CCapture::GetSaveJPGLumFactor()
{
        long result;
        InvokeHelper(0x15, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetSaveJPGLumFactor(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x15, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
```

```
                nNewValue);
}

CString CCapture::GetPICPassword()
{
        CString result;
        InvokeHelper(0x16, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, NULL);
        return result;
}

void CCapture::SetPICPassword(LPCTSTR lpszNewValue)
{
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x16, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                lpszNewValue);
}

BOOL CCapture::GetAutoSave()
{
        BOOL result;
        InvokeHelper(0x17, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

void CCapture::SetAutoSave(BOOL bNewValue)
{
        static BYTE parms[] =
                VTS_BOOL;
        InvokeHelper(0x17, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                bNewValue);
}

long CCapture::GetInterval()
{
        long result;
        InvokeHelper(0x18, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetInterval(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x18, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                nNewValue);
}

long CCapture::GetPreviewRate()
{
        long result;
        InvokeHelper(0x19, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetPreviewRate(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x19, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                nNewValue);
}

long CCapture::GetCaptionLeft()
```

```
{
        long result;
        InvokeHelper(0x1a, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetCaptionLeft(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x1a, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 nNewValue);
}

long CCapture::GetCaptionTop()
{
        long result;
        InvokeHelper(0x1b, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetCaptionTop(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x1b, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 nNewValue);
}

long CCapture::GetCaptionWidth()
{
        long result;
        InvokeHelper(0x1c, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetCaptionWidth(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x1c, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 nNewValue);
}

long CCapture::GetCaptionHeight()
{
        long result;
        InvokeHelper(0x1d, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetCaptionHeight(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x1d, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 nNewValue);
}

BOOL CCapture::GetShadowText()
{
        BOOL result;
        InvokeHelper(0x1e, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
```

```
}

void CCapture::SetShadowText(BOOL bNewValue)
{
        static BYTE parms[] =
                VTS_BOOL;
        InvokeHelper(0x1e, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                bNewValue);
}

BOOL CCapture::GetClipCaption()
{
        BOOL result;
        InvokeHelper(0x1f, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

void CCapture::SetClipCaption(BOOL bNewValue)
{
        static BYTE parms[] =
                VTS_BOOL;
        InvokeHelper(0x1f, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                bNewValue);
}

CString CCapture::GetStreamFile()
{
        CString result;
        InvokeHelper(0x20, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, NULL);
        return result;
}

void CCapture::SetStreamFile(LPCTSTR lpszNewValue)
{
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x20, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                lpszNewValue);
}

void CCapture::StartCapture()
{
        InvokeHelper(0x21, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CCapture::EndCapture()
{
        InvokeHelper(0x22, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

long CCapture::GetFrameRate()
{
        long result;
        InvokeHelper(0x23, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetFrameRate(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x23, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                nNewValue);
}
```

```
BOOL CCapture::GetYield()
{
        BOOL result;
        InvokeHelper(0x24, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

void CCapture::SetYield(BOOL bNewValue)
{
        static BYTE parms[] =
                VTS_BOOL;
        InvokeHelper(0x24, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 bNewValue);
}

BOOL CCapture::GetCaptureAudio()
{
        BOOL result;
        InvokeHelper(0x25, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

void CCapture::SetCaptureAudio(BOOL bNewValue)
{
        static BYTE parms[] =
                VTS_BOOL;
        InvokeHelper(0x25, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 bNewValue);
}

long CCapture::GetTimeLimit()
{
        long result;
        InvokeHelper(0x26, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetTimeLimit(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x26, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 nNewValue);
}

long CCapture::GetFramesProcessed()
{
        long result;
        InvokeHelper(0x27, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

long CCapture::GetFramesDropped()
{
        long result;
        InvokeHelper(0x28, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

long CCapture::GetWaveSamples()
{
        long result;
        InvokeHelper(0x29, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
```

```
        return result;
}

long CCapture::GetTimeElapsed()
{
        long result;
        InvokeHelper(0x2a, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

long CCapture::GetAudioChannels()
{
        long result;
        InvokeHelper(0x2b, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetAudioChannels(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x2b, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 nNewValue);
}

long CCapture::GetAudioBits()
{
        long result;
        InvokeHelper(0x2c, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetAudioBits(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x2c, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 nNewValue);
}

long CCapture::GetAudioSampleRate()
{
        long result;
        InvokeHelper(0x2d, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetAudioSampleRate(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x2d, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 nNewValue);
}

CString CCapture::GetFTPUserName()
{
        CString result;
        InvokeHelper(0x2e, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, NULL);
        return result;
}

void CCapture::SetFTPUserName(LPCTSTR lpszNewValue)
{
```

```
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x2e, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                lpszNewValue);
}

CString CCapture::GetFTPPassword()
{
        CString result;
        InvokeHelper(0x2f, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, NULL);
        return result;
}

void CCapture::SetFTPPassword(LPCTSTR lpszNewValue)
{
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x2f, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                lpszNewValue);
}

long CCapture::GetHDib()
{
        long result;
        InvokeHelper(0x30, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

long CCapture::GetSaveJPGSubSampling()
{
        long result;
        InvokeHelper(0x31, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetSaveJPGSubSampling(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x31, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                nNewValue);
}

BOOL CCapture::GetAutoIncrement()
{
        BOOL result;
        InvokeHelper(0x32, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

void CCapture::SetAutoIncrement(BOOL bNewValue)
{
        static BYTE parms[] =
                VTS_BOOL;
        InvokeHelper(0x32, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                bNewValue);
}

long CCapture::GetHAlign()
{
        long result;
        InvokeHelper(0x33, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}
```

```
void CCapture::SetHAlign(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x33, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 nNewValue);
}

long CCapture::GetVAlign()
{
        long result;
        InvokeHelper(0x34, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
        return result;
}

void CCapture::SetVAlign(long nNewValue)
{
        static BYTE parms[] =
                VTS_I4;
        InvokeHelper(0x34, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 nNewValue);
}

CString CCapture::GetProxyServer()
{
        CString result;
        InvokeHelper(0x35, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, NULL);
        return result;
}

void CCapture::SetProxyServer(LPCTSTR lpszNewValue)
{
        static BYTE parms[] =
                VTS_BSTR;
        InvokeHelper(0x35, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 lpszNewValue);
}

void CCapture::GrabFrame()
{
        InvokeHelper(0x36, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void CCapture::SaveFrame()
{
        InvokeHelper(0x37, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

BOOL CCapture::GetOverlay()
{
        BOOL result;
        InvokeHelper(0x38, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
        return result;
}

void CCapture::SetOverlay(BOOL bNewValue)
{
        static BYTE parms[] =
                VTS_BOOL;
        InvokeHelper(0x38, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
                 bNewValue);
}
```

```
BOOL CCapture::GetFTPRename()
{
	BOOL result;
	InvokeHelper(0x39, DISPATCH_PROPERTYGET, VT_BOOL, (void*)&result, NULL);
	return result;
}

void CCapture::SetFTPRename(BOOL bNewValue)
{
	static BYTE parms[] =
		VTS_BOOL;
	InvokeHelper(0x39, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
		bNewValue);
}

long CCapture::GetResX()
{
	long result;
	InvokeHelper(0x3a, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

void CCapture::SetResX(long nNewValue)
{
	static BYTE parms[] =
		VTS_I4;
	InvokeHelper(0x3a, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
		nNewValue);
}

long CCapture::GetResY()
{
	long result;
	InvokeHelper(0x3b, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

void CCapture::SetResY(long nNewValue)
{
	static BYTE parms[] =
		VTS_I4;
	InvokeHelper(0x3b, DISPATCH_PROPERTYPUT, VT_EMPTY, NULL, parms,
		nNewValue);
}
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Thesis16Jan.rc
//
#define IDR_CNTR_INPLACE                6
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define IDD_THESIS16JAN_FORM            101
#define IDP_FAILED_TO_CREATE            102
#define IDP_SOCKETS_INIT_FAILED         104
#define IDR_MAINFRAME                   128
#define IDR_THESISTYPE                  129
#define IDD_DLG_SETTINGS                134
#define IDD_DLG_HOSTNAME                137
#define IDC_CAPTUREPRO                  1000
#define IDC_STARTVIDEO                  1001
#define IDC_LATITUDE                    1003
#define IDC_LONGITUDE                   1004
#define IDC_ALTITUDE                    1005
#define IDC_HEADING                     1006
#define IDC_SPEED                       1007
#define IDC_GET_TELEMETRY               1008
#define IDC_STOP_SENDING                1011
#define IDC_AUTO_SEND                   1012
#define IDC_SEND_MESSAGE                1017
#define IDC_MESSAGE                     1019
#define IDC_TIME                        1020
#define IDC_BTN_CAPTURE                 1021
#define IDC_GROUP                       1022
#define IDC_LABEL1                      1023
#define IDC_LABEL2                      1024
#define IDC_LABEL3                      1025
#define IDC_LABEL4                      1026
#define IDC_LABEL5                      1027
#define IDC_LABEL6                      1028
#define IDC_RD_COM_A1                   1033
#define IDC_RD_BOARD1                   1034
#define IDC_EDIT_HOSTNAME               1035
#define IDC_EDIT_GATEWAY                1036
#define IDC_RD_COM_A2                   1037
#define IDC_RD_COM_A3                   1038
#define IDC_RD_COM_A4                   1039
#define IDC_RD_COM_B1                   1040
#define IDC_RD_COM_B2                   1041
#define IDC_RD_COM_B3                   1042
#define IDC_RD_COM_B4                   1043
#define IDC_RD_COM_IP                   1044
#define IDC_RD_BOARD2                   1045
#define IDC_EDIT_DOMAIN                 1046
#define IDC_EDIT3                       1047
#define IDC_BTN_VIDEO                   1050
#define ID_CANCEL_EDIT_CNTR             32768
#define ID_EDIT_COMMUNICATIONS          32771
#define IDS_OTH_GOLD_MSGID              61204
#define IDS_OTH_GOLD_BT                 61205
#define IDS_OTH_GOLD_CTC                61206
#define IDS_OTH_GOLD_POS                61207
#define IDS_OTH_GOLD_ENDAT              61208
#define IDS_OTH_GOLD_CTC_REPORT         61209
#define IDS_STR_GET_TRACK               61210

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
```

```
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        140
#define _APS_NEXT_COMMAND_VALUE         32772
#define _APS_NEXT_CONTROL_VALUE         1051
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

APPENDIX B: TELEMETRY SIMULATOR VISUAL C++ CODE

```
// Altus.h : main header file for the ALTUS application
//

#if !defined
        (AFX_ALTUS_H__E151CBCA_971A_11D2_B74E_0040332EE393__INCLUDED_)
#define AFX_ALTUS_H__E151CBCA_971A_11D2_B74E_0040332EE393__INCLUDED_



#if _MSC_VER > 1000

#pragma once

#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////
// CAltusApp:
// See Altus.cpp for the implementation of this class
//

class CAltusApp : public CWinApp
{
public:
        CAltusApp();

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAltusApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// Implementation
        //{{AFX_MSG(CAltusApp)
        afx_msg void OnAppAbout();
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_ALTUS_H__E151CBCA_971A_11D2_B74E_0040332EE393__INCLUDED_)
```

```
// Altus.cpp : Defines the class behaviors for the application.
//
#include "stdafx.h"
#include "Altus.h"

#include "MainFrm.h"
#include "AltusDoc.h"
#include "AltusView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAltusApp

BEGIN_MESSAGE_MAP(CAltusApp, CWinApp)
        //{{AFX_MSG_MAP(CAltusApp)
        ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
        // Standard file based document commands
        ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
        ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAltusApp construction

CAltusApp::CAltusApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CAltusApp object

CAltusApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CAltusApp initialization

BOOL CAltusApp::InitInstance()
{
        if (!AfxSocketInit())
        {
                AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
                return FALSE;
        }

        // Initialize OLE libraries
        if (!AfxOleInit())
        {
                AfxMessageBox(IDP_OLE_INIT_FAILED);
                return FALSE;
        }

        AfxEnableControlContainer();

        // Standard initialization
```

```
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	// Change the registry key under which our settings are stored.
	// TODO: You should modify this string to be something appropriate
	// such as the name of your company or organization.
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	LoadStdProfileSettings();  // Load standard INI file options (including MRU)

	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CAltusDoc),
		RUNTIME_CLASS(CMainFrame),       // main SDI frame window
		RUNTIME_CLASS(CAltusView));
	pDocTemplate->SetContainerInfo(IDR_CNTR_INPLACE);
	AddDocTemplate(pDocTemplate);

	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Dispatch commands specified on the command line
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	// The one and only window has been initialized, so show and update it.
	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();

	return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);
		// DDX/DDV support
```

```
        //}}AFX_VIRTUAL
// Implementation
protected:
        //{{AFX_MSG(CAboutDlg)
                // No message handlers
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
        //{{AFX_DATA_INIT(CAboutDlg)
        //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAboutDlg)
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
        //{{AFX_MSG_MAP(CAboutDlg)
                // No message handlers
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CAltusApp::OnAppAbout()
{
        CAboutDlg aboutDlg;
        aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CAltusApp message handlers
```

```
// AltusDoc.h : interface of the CAltusDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_ALTUSDOC_H__E151CBD0_971A_11D2_B74E_0040332EE393__INCLUDED_)
#define AFX_ALTUSDOC_H__E151CBD0_971A_11D2_B74E_0040332EE393__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CAltusDoc : public COleDocument
{
protected: // create from serialization only
        CAltusDoc();
        DECLARE_DYNCREATE(CAltusDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAltusDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CAltusDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CAltusDoc)
                // NOTE - the ClassWizard will add and remove member
                // functions here.
                // DO NOT EDIT what you see in these blocks of generated
                // code !
                //}}AFX_MSG
                DECLARE_MESSAGE_MAP()
        };
/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif // !defined(AFX_ALTUSDOC_H__E151CBD0_971A_11D2_B74E_0040332EE393__INCLUDED_)
```

```
// AltusDoc.cpp : implementation of the CAltusDoc class
//

#include "stdafx.h"
#include "Altus.h"

#include "AltusDoc.h"
#include "CntrItem.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAltusDoc

IMPLEMENT_DYNCREATE(CAltusDoc, COleDocument)

BEGIN_MESSAGE_MAP(CAltusDoc, COleDocument)
        //{{AFX_MSG_MAP(CAltusDoc)
                // NOTE - the ClassWizard will add and remove mapping
                // macros here.
                //    DO NOT EDIT what you see in these blocks of generated
                //code!
                //}}AFX_MSG_MAP
        // Enable default OLE container implementation
        ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, COleDocument::OnUpdatePasteMenu)
        ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE_LINK, COleDocument::OnUpdatePasteLinkMenu)
        ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_CONVERT, COleDocument::OnUpdateObjectVerbMenu)
        ON_COMMAND(ID_OLE_EDIT_CONVERT, COleDocument::OnEditConvert)
        ON_UPDATE_COMMAND_UI(ID_OLE_EDIT_LINKS, COleDocument::OnUpdateEditLinksMenu)
        ON_COMMAND(ID_OLE_EDIT_LINKS, COleDocument::OnEditLinks)
        ON_UPDATE_COMMAND_UI_RANGE(ID_OLE_VERB_FIRST, ID_OLE_VERB_LAST,
COleDocument::OnUpdateObjectVerbMenu)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAltusDoc construction/destruction

CAltusDoc::CAltusDoc()
{
        // Use OLE compound files
        EnableCompoundFile();

}

CAltusDoc::~CAltusDoc()
{
}

BOOL CAltusDoc::OnNewDocument()
{
        if (!COleDocument::OnNewDocument())
                return FALSE;

        return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CAltusDoc serialization
```

```
void CAltusDoc::Serialize(CArchive& ar)
{
        if (ar.IsStoring())
        {
        }
        else
        {
        }

        // Calling the base class COleDocument enables serialization
        //  of the container document's COleClientItem objects.
        COleDocument::Serialize(ar);
}

/////////////////////////////////////////////////////////////////////////////
// CAltusDoc diagnostics

#ifdef _DEBUG
void CAltusDoc::AssertValid() const
{
        COleDocument::AssertValid();
}

void CAltusDoc::Dump(CDumpContext& dc) const
{
        COleDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CAltusDoc commands
```

```
// AltusView.h : interface of the CAltusView class
//
/////////////////////////////////////////////////////////////////////////////
#include "Structures.h"       // Added by ClassView
#include "serial.h"

#if !defined(AFX_ALTUSVIEW_H__E151CBD2_971A_11D2_B74E_0040332EE393__INCLUDED_)
#define AFX_ALTUSVIEW_H__E151CBD2_971A_11D2_B74E_0040332EE393__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CAltusCntrItem;

class CAltusView : public CFormView
{
private:
        double tTime;
        int m_nTimer;
        bool m_timerset;
        CSerial serPort;

protected: // create from serialization only
        CAltusView();
        DECLARE_DYNCREATE(CAltusView)

public:
        //{{AFX_DATA(CAltusView)
        enum { IDD = IDD_ALTUS_FORM };
        double  m_LAT;
        double  m_LONG;
        float   m_SPEED;
        float   m_CSE;
        CString m_datastream;
        float   m_ALT;
        //}}AFX_DATA

// Attributes
public:
        CAltusDoc* GetDocument();
        // m_pSelection holds the selection to the current
        // CAltusCntrItem.
        // For many applications, such a member variable isn't adequate
        // to represent a selection, such as a multiple selection or a
        // selection of objects that are not CAltusCntrItem objects.
        // This selection mechanism is provided just to help you get
        // started.

        CAltusCntrItem* m_pSelection;

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAltusView)

        public:
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);
                // DDX/DDV support
        virtual void OnInitialUpdate();
```

```
            // called first time after construct
        virtual BOOL IsSelected(const CObject* pDocItem) const;
            // Container support
        //}}AFX_VIRTUAL
// Implementation
public:
        posit sPosit;
        virtual ~CAltusView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CAltusView)
        afx_msg void OnDestroy();
        afx_msg void OnSetFocus(CWnd* pOldWnd);
        afx_msg void OnSize(UINT nType, int cx, int cy);
        afx_msg void OnInsertObject();
        afx_msg void OnCancelEditCntr();
        afx_msg void OnTimer(UINT nIDEvent);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in AltusView.cpp
inline CAltusDoc* CAltusView::GetDocument()
   { return (CAltusDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif // !defined(AFX_ALTUSVIEW_H__E151CBD2_971A_11D2_B74E_0040332EE393__INCLUDED_)
```

```
// AltusView.cpp : implementation of the CAltusView class
//

#include "stdafx.h"
#include "Altus.h"
#include "AltusDoc.h"
#include "CntrItem.h"
#include "AltusView.h"
#include "Structures.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAltusView

IMPLEMENT_DYNCREATE(CAltusView, CFormView)

BEGIN_MESSAGE_MAP(CAltusView, CFormView)
        //{{AFX_MSG_MAP(CAltusView)
        ON_WM_DESTROY()
        ON_WM_SETFOCUS()
        ON_WM_SIZE()
        ON_COMMAND(ID_OLE_INSERT_NEW, OnInsertObject)
        ON_COMMAND(ID_CANCEL_EDIT_CNTR, OnCancelEditCntr)
        ON_WM_TIMER()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAltusView construction/destruction

CAltusView::CAltusView()
        : CFormView(CAltusView::IDD)
{
        //{{AFX_DATA_INIT(CAltusView)
        m_LAT = 0.0f;
        m_LONG = 0.0f;
        m_SPEED = 0.0f;
        m_CSE = 0.0f;
        m_datastream = _T("");
        m_ALT = 0.0f;
        //}}AFX_DATA_INIT
        m_pSelection = NULL;


}

CAltusView::~CAltusView()
{
}

void CAltusView::DoDataExchange(CDataExchange* pDX)
{
        CFormView::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAltusView)
        DDX_Text(pDX, IDC_EDIT_LAT, m_LAT);
        DDX_Text(pDX, IDC_EDIT_LONG, m_LONG);
        DDX_Text(pDX, IDC_EDIT_SPEED, m_SPEED);
        DDX_Text(pDX, IDC_EDIT_CSE, m_CSE);
        DDX_Text(pDX, IDC_EDIT_STREAM, m_datastream);
```

```
        DDX_Text(pDX, IDC_ALTITUDE, m_ALT);
        //}}AFX_DATA_MAP
}

BOOL CAltusView::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return CFormView::PreCreateWindow(cs);
}

void CAltusView::OnInitialUpdate()
{
        m_timerset = false;
        CFormView::OnInitialUpdate();
        GetParentFrame()->RecalcLayout();
        ResizeParentToFit();



        // TODO: remove this code when final selection model code is
        // written
        m_pSelection = NULL;    // initialize selection
}

void CAltusView::OnDestroy()
{
        // Deactivate the item on destruction; this is important
        // when a splitter view is being used.
   CFormView::OnDestroy();
   COleClientItem* pActiveItem = GetDocument()->
   GetInPlaceActiveItem(this);
      if (pActiveItem != NULL && pActiveItem->GetActiveView() == this)
   {
      pActiveItem->Deactivate();
      ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);
   }
}


/////////////////////////////////////////////////////////////////////////////
// OLE Client support and commands

BOOL CAltusView::IsSelected(const CObject* pDocItem) const
{
        // The implementation below is adequate if your selection
        // consists of only CAltusCntrItem objects.  To handle different
        // selection mechanisms, the implementation here should be
        // replaced.
        return pDocItem == m_pSelection;
}

void CAltusView::OnInsertObject()
{
        // Invoke the standard Insert Object dialog box to obtain
        // information for new CAltusCntrItem object.
        COleInsertDialog dlg;
        if (dlg.DoModal() != IDOK)
                return;

        BeginWaitCursor();

        CAltusCntrItem* pItem = NULL;
```

```
        TRY
        {
                // Create new item connected to this document.
                CAltusDoc* pDoc = GetDocument();
                ASSERT_VALID(pDoc);
                pItem = new CAltusCntrItem(pDoc);
                ASSERT_VALID(pItem);

                // Initialize the item from the dialog data.
                if (!dlg.CreateItem(pItem))
                        AfxThrowMemoryException();  // any exception will do
                ASSERT_VALID(pItem);

        if (dlg.GetSelectionType() == COleInsertDialog::createNewItem)
                        pItem->DoVerb(OLEIVERB_SHOW, this);

                ASSERT_VALID(pItem);
                m_pSelection = pItem;
                pDoc->UpdateAllViews(NULL);
        }
        CATCH(CException, e)
        {
                if (pItem != NULL)
                {
                        ASSERT_VALID(pItem);
                        pItem->Delete();
                }
                AfxMessageBox(IDP_FAILED_TO_CREATE);
        }
        END_CATCH

        EndWaitCursor();
}

// The following command handler provides the standard keyboard
//  user interface to cancel an in-place editing session.  Here,
//  the container (not the server) causes the deactivation.
void CAltusView::OnCancelEditCntr()
{
        // Close any in-place active item on this view.
        COleClientItem* pActiveItem = GetDocument()->
        GetInPlaceActiveItem(this);
        if (pActiveItem != NULL)
        {
                pActiveItem->Close();
        }
        ASSERT(GetDocument()->GetInPlaceActiveItem(this) == NULL);
}

// Special handling of OnSetFocus and OnSize are required for a
// container when an object is being edited in-place.

void CAltusView::OnSetFocus(CWnd* pOldWnd)
{
        COleClientItem* pActiveItem = GetDocument()->
                GetInPlaceActiveItem(this);
        if (pActiveItem != NULL &&
        pActiveItem->GetItemState() == COleClientItem::activeUIState)
                {
                // need to set focus to this item if it is in the same view
                CWnd* pWnd = pActiveItem->GetInPlaceWindow();
                if (pWnd != NULL)
                {
                        pWnd->SetFocus();   // don't call the base class
```

```
                        return;
                }
        }

        CFormView::OnSetFocus(pOldWnd);
}

void CAltusView::OnSize(UINT nType, int cx, int cy)
{
        CFormView::OnSize(nType, cx, cy);
        COleClientItem* pActiveItem = GetDocument()->
                GetInPlaceActiveItem(this);
        if (pActiveItem != NULL)
                pActiveItem->SetItemRects();
}

/////////////////////////////////////////////////////////////////////////////
// CAltusView diagnostics

#ifdef _DEBUG
void CAltusView::AssertValid() const
{
        CFormView::AssertValid();
}

void CAltusView::Dump(CDumpContext& dc) const
{
        CFormView::Dump(dc);
}

CAltusDoc* CAltusView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CAltusDoc)));
        return (CAltusDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CAltusView message handlers

void CAltusView::OnButton1()
{
        if (!serPort.OpenPort())
        {
                MessageBox("Error opening port...");
        }
        else
        {
                MessageBox("Port opened");
        }

}

void CAltusView::OnButton2()
{
        if (!serPort.ClosePort())
        {
                MessageBox("Error Closing port...");
        }
}

void CAltusView::OnButton3()
{
```

```
        if (!serPort.WriteData())
        {
                MessageBox("Error Writing Data...");
        }
}

void CAltusView::OnButton4()
{

        if (!m_timerset)
        {
                m_nTimer = SetTimer(1,1000, NULL);
                ASSERT (m_nTimer !=0);
                m_timerset = true;
        }
        else
        {
                KillTimer(1);
                m_timerset = false;
        }

}

void CAltusView::OnTimer(UINT nIDEvent)
{
        serPort.OpenPort();

        UpdateData(true);

        sPosit.gLat = m_LAT;
        sPosit.gLong = m_LONG;
        sPosit.gSpeed = m_SPEED;
        sPosit.gCourse = m_CSE;
        sPosit.gAltitude = m_ALT;

        m_datastream = serPort.BuildDataSet(&sPosit);


        m_LAT = sPosit.gLat;
        m_LONG = sPosit.gLong;

        UpdateData(false);

        serPort.ClosePort();
}
```

```
// CntrItem.h : interface of the CAltusCntrItem class
//

#if !defined(AFX_CNTRITEM_H__E151CBD4_971A_11D2_B74E_0040332EE393__INCLUDED_)
#define AFX_CNTRITEM_H__E151CBD4_971A_11D2_B74E_0040332EE393__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CAltusDoc;
class CAltusView;

class CAltusCntrItem : public COleClientItem
{
	DECLARE_SERIAL(CAltusCntrItem)

// Constructors
public:
	CAltusCntrItem(CAltusDoc* pContainer = NULL);
		// Note: pContainer is allowed to be NULL to enable IMPLEMENT_SERIALIZE.
		//  IMPLEMENT_SERIALIZE requires the class have a
		//  constructor with zero arguments.  Normally, OLE
		//  items are constructed with a non-NULL document pointer.

// Attributes
public:
	CAltusDoc* GetDocument()
		{ return (CAltusDoc*)COleClientItem::GetDocument(); }
	CAltusView* GetActiveView()
		{ return (CAltusView*)COleClientItem::GetActiveView(); }

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAltusCntrItem)
	public:
	virtual void OnChange(OLE_NOTIFICATION wNotification, DWORD dwParam);
	virtual void OnActivate();
	protected:
	virtual void OnGetItemPosition(CRect& rPosition);
	virtual void OnDeactivateUI(BOOL bUndoable);
	virtual BOOL OnChangeItemPosition(const CRect& rectPos);
	//}}AFX_VIRTUAL

// Implementation
public:
	~CAltusCntrItem();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif
	virtual void Serialize(CArchive& ar);
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif // !defined(AFX_CNTRITEM_H__E151CBD4_971A_11D2_B74E_0040332EE393__INCLUDED_)
```

```
// CntrItem.cpp : implementation of the CAltusCntrItem class
//

#include "stdafx.h"
#include "Altus.h"
#include "AltusDoc.h"
#include "AltusView.h"
#include "CntrItem.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAltusCntrItem implementation

IMPLEMENT_SERIAL(CAltusCntrItem, COleClientItem, 0)

CAltusCntrItem::CAltusCntrItem(CAltusDoc* pContainer)
        : COleClientItem(pContainer)
{
}

CAltusCntrItem::~CAltusCntrItem()
{
}

void CAltusCntrItem::OnChange(OLE_NOTIFICATION nCode, DWORD dwParam)
{
        ASSERT_VALID(this);

        COleClientItem::OnChange(nCode, dwParam);

        // When an item is being edited (either in-place or fully open)
        //  it sends OnChange notifications for changes in the state of
        //  the item or visual appearance of its content.

        GetDocument()->UpdateAllViews(NULL);
                // for now just update ALL views/no hints
}

BOOL CAltusCntrItem::OnChangeItemPosition(const CRect& rectPos)
{
        ASSERT_VALID(this);

        // During in-place activation CAltusCntrItem::OnChangeItemPosition
        //  is called by the server to change the position of the in-
        //  place window.  Usually, this is a result of the data in the //  server
        //  document changing such that the extent has changed or as a
        //  result of in-place resizing.
        //
        // The default here is to call the base class, which will call
        //  COleClientItem::SetItemRects to move the item
        //  to the new position.

        if (!COleClientItem::OnChangeItemPosition(rectPos))
                return FALSE;

        return TRUE;
}

void CAltusCntrItem::OnGetItemPosition(CRect& rPosition)
```

```
{
        ASSERT_VALID(this);

CAltusCntrItem::GetActiveView.

        rPosition.SetRect(10, 10, 210, 210);
}

void CAltusCntrItem::OnActivate()
{
    // Allow only one inplace activate item per frame
    CAltusView* pView = GetActiveView();
    ASSERT_VALID(pView);
    COleClientItem* pItem = GetDocument()->
                GetInPlaceActiveItem(pView);
                if (pItem != NULL && pItem != this)
                 pItem->Close();
    COleClientItem::OnActivate();
    }

void CAltusCntrItem::OnDeactivateUI(BOOL bUndoable)
{
        COleClientItem::OnDeactivateUI(bUndoable);

    // Hide the object if it is not an outside-in object
    DWORD dwMisc = 0;
    m_lpObject->GetMiscStatus(GetDrawAspect(), &dwMisc);
    if (dwMisc & OLEMISC_INSIDEOUT)
        DoVerb(OLEIVERB_HIDE, NULL);
}

void CAltusCntrItem::Serialize(CArchive& ar)
{
        ASSERT_VALID(this);

        // Call base class first to read in COleClientItem data.
        // Since this sets up the m_pDocument pointer returned from
        //  CAltusCntrItem::GetDocument, it is a good idea to call
        //  the base class Serialize first.
        COleClientItem::Serialize(ar);

        // now store/retrieve data specific to CAltusCntrItem
        if (ar.IsStoring())
        {
        }
        else
        {
        }
}

/////////////////////////////////////////////////////////////////////////////
// CAltusCntrItem diagnostics

#ifdef _DEBUG
void CAltusCntrItem::AssertValid() const
{
        COleClientItem::AssertValid();
}

void CAltusCntrItem::Dump(CDumpContext& dc) const
{
        COleClientItem::Dump(dc);
}
#endif
```

//

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__E151CBCE_971A_11D2_B74E_0040332EE393__INCLUDED_)
#define AFX_MAINFRM_H__E151CBCE_971A_11D2_B74E_0040332EE393__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);

        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__E151CBCE_971A_11D2_B74E_0040332EE393__INCLUDED_)
```

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Altus.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
        //{{AFX_MSG_MAP(CMainFrame)

        ON_WM_CREATE()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
        ID_SEPARATOR,           // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
                | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
                        CBRS_SIZE_DYNAMIC) ||
                        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
                {
                        TRACE0("Failed to create toolbar\n");
                        return -1;      // fail to create
                }

                if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
                {
                TRACE0("Failed to create status bar\n");
```

```
                return -1;      // fail to create
                }

        m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
        EnableDocking(CBRS_ALIGN_ANY);
        DockControlBar(&m_wndToolBar);

        return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        if( !CFrameWnd::PreCreateWindow(cs) )
                return FALSE;

        return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

```
// Serial.h
#include <assert.h>
#include <math.h>
#include "Structures.h"

struct sTelemetry
{
	float lLatitude;
	float lLongitude;
	float lAltitude;
	float lTimeOfFix;
	float lPitch;
	float lRoll;
	float lHeading;
	float gLatitude;
	float gLongitude;
	float gAltitude;
	float gTimeOfFix;
	float vgPitch;
	float vgRoll;
	float mHeading;
	float tAirspeed;
	byte discrete;
};

union uData
{
	sTelemetry telemetry;
	BYTE telemetryData[255];
};

class CSerial

{

private:
	double tTime;


	DCB dcbComm;
	HANDLE hComm;
	OVERLAPPED oComm;
	DWORD nrBytesToWrite;

	bool portOpened;
	bool fSuccess;

	uData outData;

	BYTE outBuff[255];

public:

	CSerial()
	{
		portOpened = FALSE;
	}

	bool OpenPort()
	{
		if (!portOpened)
		{
			hComm = CreateFile("COM3",
				GENERIC_READ |
```

```
                        GENERIC_WRITE,
                        0,
                        NULL,
                        OPEN_EXISTING,
                        NULL,
                        NULL);

                if (hComm == INVALID_HANDLE_VALUE)
                {
                // There was an error opening the port.
                        return FALSE;
                }
                // Configure the DCB
                if (!GetCommState(hComm, &dcbComm))
                {
                MessageBox(NULL, "Error in GetCommState", NULL, NULL);
                }
                dcbComm.BaudRate = CBR_19200;
                dcbComm.fBinary = TRUE;
                dcbComm.ByteSize = 8;
                dcbComm.Parity = NOPARITY;
                dcbComm.StopBits = ONESTOPBIT;
                dcbComm.EvtChar = 0x10;

                if (!SetCommState(hComm, &dcbComm))
                {
                MessageBox(NULL, "Error in SetCommState", NULL, NULL);
                }
                // Set the communications mask

                if (!SetCommMask (hComm, EV_RXCHAR | EV_RXFLAG))
                {
                MessageBox(NULL,"Error in SetCommMask", NULL, NULL);
                }
                oComm.Offset = 0;
                oComm.OffsetHigh = 0;
                oComm.hEvent = CreateEvent (NULL, TRUE, FALSE, NULL);
                assert (oComm.hEvent);
                portOpened = TRUE;
                return TRUE;
        }
        return FALSE;
}
bool ClosePort()
{

        if (portOpened)
        {
                CloseHandle(hComm);
                portOpened = FALSE;
                return TRUE;
        }
        return FALSE;
}
bool ReadData()
{
}
bool WriteData()
{
        DWORD lastError;
        DWORD bytesWritten;

        if (!portOpened)
        {
```

```
                return FALSE;
        }
        if (!WriteFile (hComm,
                                outBuff,
                                nrBytesToWrite,
                                &bytesWritten,
                                &oComm))
        {
                lastError = GetLastError();
        }
        return TRUE;
}

CString BuildDataSet(posit* inPosit)
{
        CString newHex = "";
        CString junque;
        float crse = inPosit->gCourse;
        double xini = inPosit->gLat;
        double yini = inPosit->gLong;
        double distance = inPosit->gSpeed / double(3600 * 60);

        double Lat;
        double Long;

        float cCourse;

                cCourse = 90 - crse;

        if (cCourse < 0.0F) cCourse += 360.0;

        double factor = 0.017453293F;


        double x = double(distance * sin(cCourse * factor));
        double y = double(distance * cos(cCourse * factor));

                Lat = (x + xini);
                Long = (y + yini);
                outData.telemetry.lLatitude = float(Lat * factor);
                outData.telemetry.lLongitude = float(Long * factor);
                outData.telemetry.lAltitude = inPosit->gAltitude;
                outData.telemetry.lTimeOfFix = 1112.30F;
                outData.telemetry.lPitch = 0.0F;
                outData.telemetry.lRoll = 0.0F;
                outData.telemetry.lHeading = crse;
                outData.telemetry.gLatitude = 0.0F;
                outData.telemetry.gLongitude = 0.0F;
                outData.telemetry.gAltitude = inPosit->gAltitude;
                outData.telemetry.gTimeOfFix = 1112.30F;
                outData.telemetry.vgPitch = 0.0F;
                outData.telemetry.vgRoll = 0.0F;
                outData.telemetry.mHeading = 90.0F;
                outData.telemetry.tAirspeed = inPosit->gSpeed;
                outData.telemetry.discrete = 0xff;

                inPosit->gLat = Lat;
                inPosit->gLong = Long;


                int byteCounter;
                int counterOffset = 2;
                byte testByte;
                byte checkByte = 0x10;
```

```
                    outBuff[0] = 0x10;
                    outBuff[1] = 0x1;

                    junque.Empty();

                    for (byteCounter = 0; byteCounter < 61; byteCounter++)
                    {
                            testByte = outData.telemetryData[byteCounter];
                            outBuff[byteCounter + counterOffset] =
                                    outData.telemetryData[byteCounter];
                            newHex.Format("%X", testByte);
                            junque += newHex;

                            if (testByte == checkByte)
                            {
                                    counterOffset = counterOffset + 1;
                                    outBuff[byteCounter + counterOffset] =
                                            outData.telemetryData[byteCounter];
                            }

                    }
                    outBuff[byteCounter + counterOffset] = 0x10;
                    outBuff[byteCounter + counterOffset + 1] = 0x3;
                    nrBytesToWrite = byteCounter + counterOffset + 2;
                    // Send the data
                    WriteData();

            return junque;
    }
};
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__E151CBCC_971A_11D2_B74E_0040332EE393__INCLUDED_)
#define AFX_STDAFX_H__E151CBCC_971A_11D2_B74E_0040332EE393__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN          // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxole.h>         // MFC OLE classes
#include <afxodlgs.h>       // MFC OLE dialog classes
#include <afxdisp.h>        // MFC Automation classes

#ifndef _AFX_NO_DB_SUPPORT
#include <afxdb.h>                  // MFC ODBC database classes
#endif // _AFX_NO_DB_SUPPORT

#ifndef _AFX_NO_DAO_SUPPORT
#include <afxdao.h>                 // MFC DAO database classes
#endif // _AFX_NO_DAO_SUPPORT

#include <afxdtctl.h>         // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                 // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxsock.h>          // MFC socket extensions

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations
// immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__E151CBCC_971A_11D2_B74E_0040332EE393__INCLUDED_)
```

```
// stdafx.cpp : source file that includes just the standard includes
//      Altus.pch will be the pre-compiled header
//      stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Altus.rc
//
#define IDR_CNTR_INPLACE                6
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define IDD_ALTUS_FORM                  101
#define IDP_FAILED_TO_CREATE            102
#define IDP_SOCKETS_INIT_FAILED         104
#define IDR_MAINFRAME                   128
#define IDR_ALTUSTYPE                   129
#define IDC_BUTTON1                     1000
#define IDC_BUTTON2                     1001
#define IDC_BUTTON3                     1002
#define IDC_BUTTON4                     1003
#define IDC_EDIT_LAT                    1004
#define IDC_EDIT_LONG                   1005
#define IDC_EDIT_CSE                    1006
#define IDC_EDIT_SPEED                  1007
#define IDC_EDIT_STREAM                 1008
#define IDC_ALTITUDE                    1009
#define ID_CANCEL_EDIT_CNTR             32768

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        130
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1010
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

APPENDIX C: UTI²PS IMAGERY DATABASE DEFINITION

```
/* Microsoft SQL Server - Scripting                */
/* Server: RATBERT                                 */
/* Database: imagery                               */
/* Creation Date 3/18/99 4:31:24 PM                */

set quoted_identifier on
GO

/****** Object:  Table dbo.tblAltusImages    Script Date: 3/18/99
4:31:26 PM ******/
if exists (select * from sysobjects where id =
object_id('dbo.tblAltusImages') and sysstat & 0xf = 3)
      drop table "dbo"."tblAltusImages"
GO

/****** Object:  Table dbo.tblImages    Script Date: 3/18/99 4:31:26 PM
******/
if exists (select * from sysobjects where id =
object_id('dbo.tblImages') and sysstat & 0xf = 3)
      drop table "dbo"."tblImages"
GO

/****** Object:  Table dbo.tblAltusImages    Script Date: 3/18/99
4:31:26 PM ******/
CREATE TABLE "dbo"."tblAltusImages" (
      "fldImageName" char (255) NOT NULL ,
      "fldDateTime" "smalldatetime" NOT NULL ,
      "fldLatitude" char (255) NULL ,
      "fldLongitude" char (255) NULL ,
      "fldAltitude" char (255) NULL ,
      "fldRemarks" char (255) NULL ,
      "fldImage" "image" NULL ,
      CONSTRAINT "PK___1__10" PRIMARY KEY  CLUSTERED
      (
            "fldDateTime"
      )
)
GO

/****** Object:  Table dbo.tblImages    Script Date: 3/18/99 4:31:26 PM
******/
CREATE TABLE "dbo"."tblImages" (
      "fldImageName" char (50) NOT NULL ,
      "fldDateTime" "datetime" NOT NULL ,
      "fldLatitude" char (6) NOT NULL ,
      "fldLongitude" char (7) NOT NULL ,
      "fldAltitude" char (6) NOT NULL ,
      "fldRemarks" char (255) NULL ,
      "fldTrack" char (6) NOT NULL
)
GO
```

APPENDIX D: UTI²PS WEB PAGE HTML CODE

```
<html>
<head><script language="JavaScript">

<!--//
function OpenPopup() {
   openpopup=window.open("realtime.htm","popupwin","width=340,height=260,
                                 left=00,top=30,resizable=no");
   openpopup.opener.name = "opener";
}


function OpenPopup2() {
   openpopup2 =window.open("allimages.asp","new","width=500,height=600,
                               left=00,top=00,resizable=yes, scrollbars=yes,
                               refresh=yes");
   openpopup2.opener.name = "opener2";
}
//-->
</script>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Home Page</title>
<meta HTTP-EQUIV="refresh" CONTENT="10; url=default.asp">
<meta name="Microsoft Border" content="t, default">
</head>
<font name="tahoma">

<body bgcolor="#C0C0C0">

<form>
  <p><!--#include file="adovbs.inc"--> <%
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "altus","webuser",""

Set rsEvents = Conn.Execute("SELECT * from tblImages ORDER BY fldDateTime
DESC")
Response.Write "<center>"
If rsEvents.EOF then

   Response.Write "<p>No current images, try again later</p>"

Else

   Response.Write "<p><font face=Tahoma color=#000080>
          Most Recent Track</font></p>"
   Response.Write "<p>"
   Response.Write "<img src=/images/" &
          rsEvents.Fields.Item("fldImageName").Value & ">"
   Response.Write "<p><font face=Tahoma>"
   Response.Write rsEvents.Fields.Item("fldDateTime") & "</font>"
End if
Response.Write "</center>"
%></p>
</form>
```

```
<form method="GET" action="allimages.asp">
  <div align="center"><center><p><input type="button" value="Live Video"
name="video"
  onClick="OpenPopup()"> <input type="button" value="Image Database"
name="still"
  onClick="OpenPopup2()"> </p>
  </center></div>
</form>

</font>
</body>
</html>
```

HTML Source for ALLIMAGES.ASP

```
html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<title>Imagery Database</title>
<meta HTTP-EQUIV="refresh">
<meta name="Microsoft Border" content="t, default">
</head>

<body bgcolor="#C0C0C0">

<p> </p>

<p><!--#include file="adovbs.inc"--> </p>
<div align="center"><%
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "altus","webuser",""
Set rsEvents = Conn.Execute
       ("SELECT * from tblImages ORDER BY fldDateTime DESC")

If rsEvents.EOF Then
       Response.Write "Sorry, no images at this time..."
Else
%>
<div align="center"><center>

<table CELLSPACING="10" BORDER="1">
<%
   While Not rsEvents.EOF
       Response.Write "<tr><td rowspan = 6><img src=/images/" &
                 rsEvents.Fields.Item("fldImageName").Value & "</td>"
       Response.Write "<td><font face=tahoma font color=#FF0000>" &
                 rsEvents.Fields.Item("fldTrack").Value & "</font></td></tr>"
       Response.Write "<tr><td><font face=tahoma>" &
                 FormatDateTime(rsEvents.Fields.Item("fldDateTime").Value, 2) &
                 "</font></td></tr>"
       Response.Write "<tr><td><font face=tahoma>" &
                 FormatDateTime(rsEvents.Fields.Item("fldDateTime").Value, 4) &
                 "</font></td></tr>"
       Response.Write "<tr><td><font face=tahoma>" &
                 rsEvents.Fields.Item("fldLatitude").Value & "</font></td></tr>"
       Response.Write "<tr><td><font face=tahoma>" &
                 rsEvents.Fields.Item("fldLongitude").Value &
                 "</font></td></tr>"
       Response.Write "<tr><td><font face=tahoma>" &
                 rsEvents.Fields.Item("fldAltitude").Value & "</font></td></tr>"
       rsEvents.MoveNext
   Wend
End If
%>
</table>
</center></div></div>
</body>
</html>
```

APPENDIX E: GCCS RELAY VISUAL BASIC CODE

```
' frmGCCSRelay

Private Sub cmdSelPort_Click()
      frmConfig.Show
      MSComm.Handshaking = comNone
      MSComm.Settings = "19200,N,8,1"
End Sub

Private Sub Form_Load()
      gccs.Protocal = sktTCPProtocol
      gccs.LocalPort = 2071
      gccs.Listen
End Sub

Private Sub gccs_Close()
      gccs.Close
      gccs.Protocol = sktTCPProtocol
      gccs.LocalPort = 2071
      gccs.Listen
End Sub

Private Sub gccs_ConnectionRequest(ByVal requestID As Long)
      If gccs.State <> sckClosed Then gccs.Close
      gccs.Accept requestID
End Sub

Private Sub gccs_DataArrival (ByVal bytesTotal As Long)
      Dim msgData as String
      GoldMessage.Text = ""
      gccs.GetData msgData, vbString, bytesTotal
      GoldMessage.Text = GoldMessage & msgData
      On Error GoTo InvalidCommPort
      MSComm.PortOpen = True
      MSComm.Output = GoldMessage.Text
      MSComm.PortOpen = False
      On Error GoTo 0
      Exit Sub
InvalidCommPort:
      errorValue = MsgBox("Invalid Comm Port", vbOKOnly + vbCritical)
      frmConfig.Show
End Sub
Private Sub gccs_Error(ByVal Number As Integer, Description As String,
                  ByVal Scode As Long, ByVal Source As String,
                  ByVal HelpFile As String,
                  ByVal HelpContext As Long,
                  CancelDisplay As Boolean)
      MsgBox (Description)
End Sub
```

```
'frmConfig

Private Sub_FormLoad()
      frmGCCSRelay.MSComm.Port = 1
      Option1(0).Value = True
      With Label1
            .Caption = "GCCS Gateway Setting:" & Chr(13) & Chr(13)
            .Caption = .Caption & "19200 baud" & Chr(13)
            .Caption = .Caption & "8 data bits" & Chr(13)
            .Caption = .Caption & "1 stop bit" & Chr(13)
            .Caption = .Caption & "no parity" & Chr(13)
            .Caption = .Caption & "no handshaking"
      End With
End Sub

Private Sub Option1_Click(Index As Integer)
      frmGCCSRelay.MSComm.CommPort =  (Index + 1)
End Sub
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd
   Monterey, CA 93943-5101

3. Dean Netzer 1
   Research Office, Code 09
   Naval Postgraduate School
   Monterey, CA 93943-5138

4. Office of Chief of Naval Operations 1
   Attn. Dr. Frank Shoup
   Expeditionary Warfare Division (N85)
   2000 Navy Pentagon
   Washington, DC 20350-2000

5. Professor John Osmundson 1
   C3 Academic Group, Code CC/OS
   Naval Postgraduate School
   Monterey, CA 93943-5103

6. Professor Gary Porter 2
   C3 Academic Group, Code CC/PO
   Naval Postgraduate School
   Monterey, CA 93943-5103

7. Paul Finn 1
   CIRPAS Program Manger
   3240 Imjin Rd., Hanger #510
   Marina, CA 93933

8. Mike Duncan 1
   CIRPAS Operations Engineer
   3240 Imjin Rd., Hanger #510
   Marina, CA 93933

9. LCDR Andrew Cameron 2
   76 Brittany Lane
   Stafford, VA 22554-7687

10. LT John Cherry 2
    17030 I Street
    Omaha, NE 68135